

Jan de Jane









# NAVAL POSTGRADUATE SCHOOL

Monterey, California



# THESIS

PETRI-NET SIMULATIONS
OF
COMMUNICATIONS NETWORKS

by

Stephen Craig Jennings

Robert John Hartel

March 1980

Thesis Advisor:

Lyle A. Cox

Approved for public release; distribution unlimited

# NAVAL POSTGRADUATE SCHOOL Monterey, California

Rear Admiral J. J. Ekelund Superintendent Jack R. Borsting Provost

Reproduction of all or part of this report is authorized.

Released as a Technical Report by:

SECURITY CLASSIFICATION OF THIS PAGE (When Date Entered)

REPORT DOCUMENTATION	READ INSTRUCTIONS BEFORE COMPLETING FORM						
NPS52-80-003	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER					
Petri-Net Simulations of Communic	S. TYPE OF REPORT & PERIOD COVERED Master's Thesis; March 1980						
7. AUTHOR(*)	8. PERFORMING ORG. REPORT NUMBER 8. CONTRACT OR GRANT NUMBER(s)						
Stephen Craig Jennings Robert John Hartel							
Naval Postgraduate School Monterey, CA 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS					
Naval Postgraduate School	March 1980						
Monterey, CA 93940	13. NUMBER OF PAGES						
TA MONITORING AGENCY NAME & ADDRESS(II dilleren	i irom Controlling Ollice)	UNCLASSIFIED					
16. DISTRIBUTION STATEMENT (of this Report)		15a. DECLASSIFICATION/DOWNGRADING					

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the enerrors entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19 KEY WORDS (Continue on reverse side il necessary and identify by block number)

Simulation Petri-Net Evaluation

20. ABSTRACT (Continue on reverse side il necessary and identify by black number)

Current technologies in the fields of telecommunications and computer processing are becoming increasingly integrated to the extent that "distributed" computer networks are assuming key roles in communications. The complex computerized systems necessary to support modern military command and control requirements are expensive. Designing such systems by trial and error is not feasible, yet no other viable alternatives exist. This thesis offers an original methodology for evaluating the predicted

THE RESERVE

1 Add to Add

the bank of the second



UNCLASSIFIED SECURITY CLASSIFICATION OF THIS PAGE/When Date Entered performance of military automated systems. Using Petri-Nets as a modeling tool, computer simulations with color graphics output are performed to demonstrate the feasibility of this approach as a systems design tool.



Approved for public release; distribution unlimited.

PETRI-NET SIMULATIONS OF COMMUNICATIONS NETWORKS

Stephen C. Jennings Captain, United States Marine Corps E.S., United States Naval Academy, 1971

Robert J. Hartel Captain, United States Army B.S., Texas A&M University, 1972

Submitted in partial fulfillment of the requirements for the degrees of

MASTER OF SCIENCE IN SYSTEMS TECHNOLOGY (C3)
(Stephen C. Jennings)

MASTER OF SCIENCE IN COMPUTER SCIENCE
(Robert J. Hertel)

from the NAVAL POSTGRADUATE SCHOOL March, 1980



#### ABSTRACT

Current technologies in the fields of telecommunications and computer processing are becoming increasingly integrated to the extent that "distributed" computer networks are assuming key roles in communications. The complex computerized systems necessary to support modern military command and control requirements are expensive. Designing such systems by trial and error is not feasible, yet no other viable alternatives exist. This thesis offers an original methodology for evaluating the predicted performance of military automated systems. Using Petri-Nets as a modeling tool, computer simulations with color graphics output are performed to demonstrate the feasibilty of this approach as a systems design tool.



# TABLE OF CONTENTS

I.	INTE	ROTUCTION	9					
II.	NTTV	VOPKS: MOTIVATION, TAXONOMY AND PEPFORMANCE	14					
	Α.	INTRODUCTION	14					
	В.	MOTIVATION FOR THE ANALYSIS OF PEPFORMANCE	14					
	С.	TAXONOMY	16					
		1. Networks	16					
		2. Computer Communication Networks	16					
		3. Further Reading	18					
	D.	PERFORMANCE PREDICTION AND MODELING	18					
	F.	SUMMARY	21					
III.	MILITARY APPLICATIONS OF DISTRIBUTED SYSTEM							
	TECH	RNCLOGY	22					
	Α.	INTRODUCTION	22					
	B.	PACKET SWITCHING	22					
	С.	THE ARPANET	25					
	D.	MILITARY APPLICATIONS	27					
	E.	THE ALOHA SYSTEM	28					
	F.	PACKET RADIO INTRODUCTION	29					
	G.	CHANNFL ACCESS SCHEMES	31					
		1. Category I	32					
		2. Category II	36					
		3. Category III	37					
	Н.	CURRENT PROGRAMS	38					
		1. PLRS/JTIDS Hybrid	39					



		2. The San Francisco Bay Experimental	
		Packet Fadic Network	39
		3. The Fort Bragg Testbed	41
	I.	FUTURE IMPLICATIONS	43
		1. Chain of Command	44
		2. Network Management	45
		3. How Much Redundancy?	46
		4. Propagation Loss Due to Higher Frequency	46
		5. Management Information Systems	47
		6. Interoperability	47
		7. Voice vs. Data Circuits	48
		8. System Cost	48
IV.	AN	INTRODUCTION TO PETRI-NETS	49
	Α.	INTRODUCTION	49
	В.	#ISTORY	49
	С.	FOW PETRI-NFTS WORK	51
	D.	A SIMPLE EXAMPLE	54
	<u> </u>	ADVANTAGES AND DISADVANTAGES	54
₹.	PETI	RI-NET SIMULATION OF COMPUTEP COMMUNICATION	
	NET	ORKS	60
	Α.	INTRODUCTION	$\epsilon e$
	В.	TYPES OF NODES	61
	С.	A SIMPLE APPLICATION EXAMPLE	62
	D.	RANDOMNESS IN PETRI-NETS	64
	E.	MEMORY STORAGE REPPESENTATION	67
	F.	SYSTEM LOAD AVERAGE	67
	G.	TIME REPRESENTATION	68



VI.	THE	EXPERIMENT 71
	Α.	rescription
	В.	RESULTS 73
VII	I. REC	OMMENDATIONS AND CONCLUSIONS
	A .	RECOMMENDATIONS
	В.	CONCLUSIONS 80
APF	PENDIX	A - USER INSTRUCTIONS FOR PETRI-NET SOFTWARE 82
	A .	INTRODUCTION 82
	З.	THE INPUT FILE 82
		1. Places £4
		2. Transitions
		3. Initial Marking εε
		4. Execution 87
	С.	TFE OUTPUT FILES 87
	D.	USFR OPTICNS
		1. Choice of Programs 88
		2. User Questions and Responses 88
		3. The Highlighting Feature91
	Ð.	UNIQUE INSTRUCTIONS FOR VEPSION 3
API	RIDNES	B - RUN23
API	PENDIX	C - RUN 03A 98
API	PENDIX	r - Rungse 99
API	SENDIX	파 - RUN 03C100
API	PENDIX	F - PUN20
API	PENDIX	G - THESIS FILES129
CON	PUTER	PROGRAM - "simulator"
CON	PUTER	PPOGRAM - "transgraph"119



COMPUTE	PROGRAM - "1	inkgraph"	• • •	 	 	 • •	• •	• • •	 .1	47
LIST OF	REFFRENCES			 	 	 			 .1	74
INITIAL	DISTRIBUTION	LIST		 	 	 			 .1	76



## I. INTRODUCTION

As military planners look forward to the design of future command, control, and communications (C3) systems, several key factors should be at the forefront of their thinking. First of all, it is readily apparent that there has been a proliferation of computer resources for military applications. Even the most "tactical" of systems today is becoming a sizeable collection of computers, databases, sensors and information-handling equipment.

Secondly, one can sense that the fields of telecommunications and computer science are becoming increasingly integrated. Although these once-separate disciplines have very different histories and traditions, they are experiencing a technological convergence which is having far-reaching implications for both the military and civilian societies. Today the concepts and techniques of computer processing have been integrated with communications to the extent that both fields share the same kind of logic, storage, switching and transmission. Because information handling systems now employ telecommunications and information in such an intimate mixture, it is difficult to distinguish what in the system is computer processing and what is communications.

A third key factor should be a realization that certain concepts of computer communication networks and distributed



system architectures offer advantages for military applications due to the potential for survivability and lack of centralization.

Computer networks are often created spontaneously by combining computers and communications. The growth of computer networks is one of the significant outcomes of the convergence of the two disciplines. An extensive array of computing resources can be connected over a wide geographic area via telecommunications channels. The potential architectures for such networks are limitless when one considers the variety of hardware, software, protocols and geographic distributions that might comprise the system.

Another key factor is that system planners need to focus their efforts upon total system integration. Tactical automated systems tend to be developed individually to meet unique mission requirements. For instance, separate systems are typically justified and developed for missions of fire control, air defense, intelligence, personnel management and logistics, etc. While these separate systems may peform satisfactorily alone, there is difficulty in providing a suitable management information system by which the overall commander or decision-maker can have access to the information in all these separate systems in a format that is easy to understand and use.

Another important consideration is that the proliferation of automated systems places a severe burden upon the communications equipment that links systems



together. There is a tendency for automated systems to be separately developed with insufficient emphasis placed upon the communications equipment that will transmit the information. In other words, the sensors and processors of a system may work splendidly, but planners must not take it for granted that the data will arrive at the correct destination in an error-free condition.

The communications equipment and channelization that carry the information must be as carefully engineered as the other components of the system. In addition, the data on the channel must be in a format that is compatible with other systems. The U.S. Army is coming to grips with the fact that if the some 50 tactical automated systems on the drawing boards were to be fielded for use at the corps level, there exists no communications equipment capable of carrying the vast volume of information these systems would generate. In addition, when the communications equipment rust operate with specifications of transmission security, jam resistance, and low probability of intercept in a severe electronic warfare threat environment, the design difficulties are considerably magnified.

Lastly, military planners should be concerned about being able to accurately predict system performance. An accurate predictor of system performance is needed for use by those in procurement duties to ensure that performance specifications given by contractors will in fact prove to be true when a new system is fielded. Managers of currently



operational systems also require this service. They are asking in the course of daily duties such questions as: That would be the effect of increased buffer space at this busy location? What would be the impact on total system performance if a particular node or link in the network is removed? Such a desire for prediction of system performance has created great emphasis upon modeling techniques and computer simulations of systems to answer these questions.

All of the above factors have given impetus to this thesis. A particular modeling tool which addresses these needs is described (the Petri-Net) and is implemented to facilitate communications network modeling.

The paper presents three primary points of original work:

- 1. It is demonstrated that automated networks can be meaningfully modeled with the use of Petri-Nets.
- 2. It is shown that Petri-Net models of networks can be adapted for effective computer execution and display on a color graphics terminal. Simulations which incorporate graphics output are more easily understood and have considerable educational value.
- 3. The results of this research indicate that the implementation of such a modeling technique in a production environment as a predictor of system performance is feasible.

The above 3 points, although successfully implemented in the Naval Postgraduate School C3 laboratory, represent a



new area of research in a preliminary stage of development.

Future investigation is required to expand and validate this approach.



### II. NETWORKS: MOTIVATION, TAXONOMY AND PERFORMANCE

#### A. INTRODUCTION

Predicting the performance of an automated network can indicate a measure of a system's effectiveness and efficiency.[3] Evaluating a system's performance is a complex task. This task often requires modeling. Many performance modifications are more suitably performed on models than the actual system, because "trial and error" production is not feasible economically. This chapter discusses the motivations for network design and the practicalities of predicting network performance.

### B. MOTIVATION FOR THE ANALYSIS OF PERFORMANCE

The magnitude of information processing in the United States is unprecedented and still growing. Computer processing and communications make a major portion of currently accessable information available to federal agencies and commercial businesses. As the country is becoming increasingly dependent on the need for information, the existence of reliable, effective computer communication networks is essertial to transport computer-processed information.

With the added cost of energy, the attractiveness of moving resources to the user via computer communications networks is apparent. [5] The performance evaluation of these networks is, therefore, a measure of the system's ability to



meet user requirements.

Adequate performance evaluation tools currently do not exist. Therefore, decision-makers responsible for selecting new systems usually rely on the designer's claims as to performance, and procure systems accordingly.

Both the civilian and military communities place very heavy demands on communication's facilities during crisis situations. Throughout history existing systems have often not been sufficient to carry such communications traffic.

[8]

The civilian community shares these problems. Many networks are engineered to carry mean traffic loads and are not planned for crisis contigencies. The commercial telephone network, for instance, is inundated with traffic Christmas and Mother's Day, severely degrading system performance. The disruptive affects of a national emergency on all types of automated networks can only be imagined.

In the Defense Communications Agency (DCA), work is presently under way to develop tools for evaluating the performance of planned computer communication retworks. This effort is underway because development of these expensive systems by "trial and error" can no longer be afforded. The necessity to have confidence in the system's ability to meet design specifications before production is essential, since normal federal procurement cycles stretch cut over eight to ten years. This confidence can be insured by utilizing effective performance evaluation tools.



#### C. TAXONOMY

A great deal of ambiguity exists in the jargon of the networking field. [7] Therefore, several recurring network descriptions are defined in this chapter to provide a consistent vocabulary. Using these definitions, network issues can then be succinctly conveyed to the reader.

## 1. Natworks

The term "network" conveys the concept of irdividualized cells and a degree of interconnectivity. A network exists for the purpose of achieving a desired objective. The individual characteristics of networks are related to network topology, hardware configurations, and software control features designed to accommodate the user's requirements.

# 2. Computer Communication Networks

A "computer communication network" is a system consisting of one or more computers and terminals, and a communications subsystem which connects them. [7] The primary purpose of this network is to facilitate the efficient flow of data, and provide the required supportive processing functions. The communications subsystem consists of transmission facilities and associated communications processors. Communications processors are computers dedicated to exclusively handling communications tasks.

The classification of computer communication networks often centers around the network topology, network connectivity, switching protocol and the degree of



implementation of system-wide control features.

The term "centralized computer communication network" is used to define a network that possesses a high degree of centralized functions. Another network classification, "distributed systems", is used to describe a low degree of centralized functions.

The differences between centralized computer communication networks and distributed systems reveal themselves in the degree to which system functions are distributed. There are no concise metrics which delineate the exact classification of a network. The interpretation is subjective.

Further elaboration on specific advantages and disadvantages of the distribution of system functions and the definition of distributable functions can be found in CYSPER [6].

Centralized computer networks essentially formalize the system control structure into pre-selected system control nodes. These nodes contain what is often referred to as the network control programs (NCP). The NCP can control the processing, database management and communications management functions. This type of networking characteristically has a low degree of fault tolerance because network direction eminates from only one node imposing network restrictions should this node be damaged.

Distributed systems, however, are typified by the distribution of system management functions. Although no



"pure" distributed systems exist in reality, the term is widely used to indicate a high degree of distributed functions within a network. This "peer" structure acts in a cooperative sense. Fouting algorithms, for example, rely on information that is "cooperatively" passed from node to neighboring node, thereby deriving information, not on a global basis but on a localized one. Distributed systems have a high degree of survivability and are, therefore, more fault tolerant than centralized computer communication networks. The advantage of distributed systems is, however, partially offset by the increased "overhead" necessary to coordinate system functions.

# 3. Further Reading

Numerous terms used in networking contribute to confusion due to the lack of an industry-wide standardized taxonomy. The intent here was to define certain keywords used throughout this thesis. Further clarification of system's taxonomy, although important, is not discussed. Taxonomical studies recommended are [2], [9] and [23].

## D. PERFORMANCE PREDICTION AND MODELING

"Performance" is defined as the degree to which the system fulfills user requirements. In terms of networks, these measures are often determined by the network's workload capacity or throughput in the sense that the network can first perform the desired functions and secondly perform with a degree of timeliness. [4]



The ideal means of measuring network performance is to extract data from the system itself. The collection of data from existing networks is often difficult. The performance testing of networks under heavy traffic loads or those operating in a degraded mode is a sensitive matter, because interference of the monitoring equipment can not be tolerated. Networks in design phases, of course, can not be measured and require alternative assessment methods.

The modeling of existing and planned networks has become an important component of performance evaluation. As an added benefit, modeling also facilitates the understanding of a system's design and interrelationships.

The modeling process can follow one of several different techniques or a combination thereof. The primary techniques are: 1) mathematical modeling and 2) simulation.

Mathematical modeling employs theories of queuing and flow by describing certain network characteristics in sets of equations. The process is, however, complex and often assumes away critical parameters. The primary disadvantage with mathematical modeling appears to be just this problem of assumptions. Too many assumptions impose an unacceptable degree of abstraction. Although the validity of mathematical modeling techniques has been confirmed [4], the methodology is often understandable only by the modeler.

Simulation, the second alternative, leads the modeler to numerous techniques. A simulation is an abstraction of concepts pertinent to the problem being studied, and upon



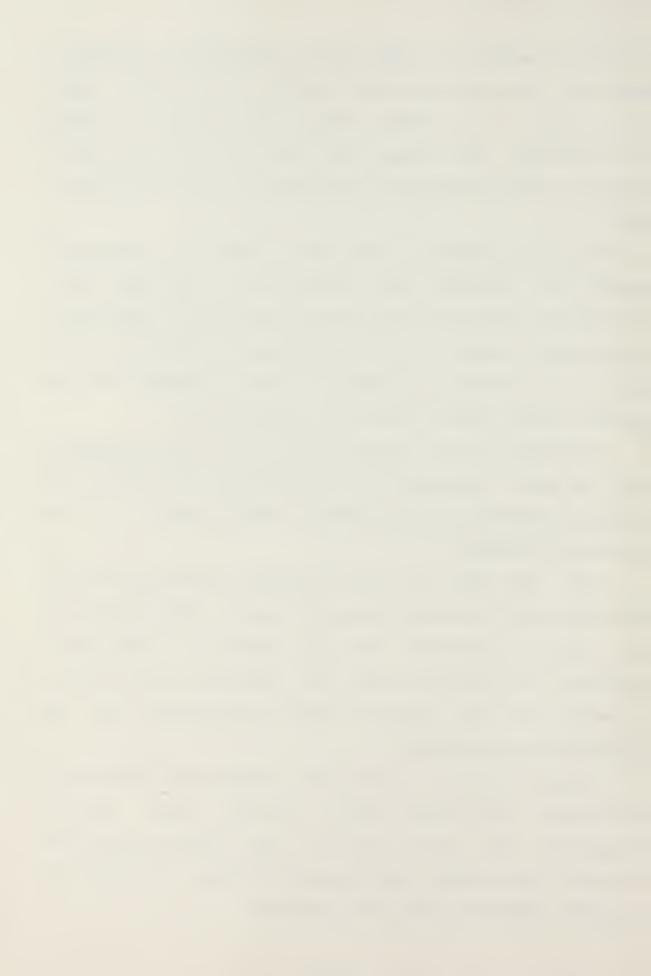
which the modeler can apply varying experimental variables. The model simulates only those features the modeler feels are relevent to the problem. Herein lies the critical danger of simulation. The danger may best be expressed by the question, Does the model bear relevancy to the real problem? [10]

The major problem of simulation, as well as analytical models, is therefore the validation of the model. Many simulation experts talk in terms of performance reliability factors but fail to state that these factors may be just derived from outputs of a model, and the closeness of the model to reality may or may not be substantiated.

The modeling process itself consists of the construction of the model, followed by the validation of the model, and finally modifications to the model based on results of the validation process.

Once the model has been defined, simulations are run with the model to evaluate network behavior. The problem is to predict performance of real networks by evaluating behavior on the network models. The simulations then provide a means by which network design deficiencies can be identified and corrected.

Another issue in simulation analysis is the area of overdesign. Simulations should indicate those systems components that do not add to the capabilities of the network. This analysis can measure the device utilization of specific components within the network.



The key measurement of a computer communication network is the network's workload capacity or throughput. Throughput is generally measured in number of message units per time period, and provides a measure of effectiveness and efficiency of the system. The parameters involved in throughput are: 1) network configuration (topology), 2) network control algorithms, and 3) network reliability.

#### E. SUMMARY

A solution to predicting performance in systems where the collection of data is difficult or the system is in design stages, is to build a model of the system. The model could then be tested over the entire sprectrum of performance specifications.

The goal of performance evaluation is the prediction of the degree to which the system fulfills the intended objectives. The major concern in computer communication networks is the degree to which the network can perform the task and the degree of efficiency with which the task is completed. A by-product of the evaluation should be the identification of areas of over and under-design. Once design failures have been identified, design tradeoff decisions can then be made. This process of performance analysis is simed at optimizing the existing or planned network's performance and ensuring that the performance meets the contracted user recuirements.



# III. MILITARY APPLICATIONS OF DISTRIBUTED SYSTEM TECHNOLOGY

### A. INTRODUCTION

This chapter discusses current military programs and research efforts that are applying the concept of automated networks to tactical missions. It is written to give the reader some background information on programs and terminology. With this background, the applicability of the simulations described in later chapters will be clearer.

## B. PACKET SWITCHING

The transmission of computer to computer digital messages has had significant impact on communications switching techniques. In fact, the concept of packet switching was invented to a large extent because of the unique requirements of computer based systems. Packet switching was designed as an alternative to circuit switching.

The circuit switching technique of older communications systems is a method of establishing a route for communications traffic whereby a complete link between the calling and receiving station is set up and maintained exclusively for the exchange of those two stations. The connection is maintained until one of the stations breaks off transmission or reception. A technique such as this tends to be wasteful in computer communications because computer communications are typically "bursty" in nature;



that is, the messages are very short in duration and require fast responses.

Packet switching is designed to make efficient use of a communications channel when the traffic is bursty. [11] In this technique messages are divided into discrete "packets." A packet is a block of information containing a fixed number of bits. Each packet contains the text of the message plus a control header. The header contains enough information (for example, source, destination, routing plan, message sequence number, etc.) to guarantee the packet will arrive at the proper destination. In addition, there will usually be some checks on each such block, so that any switch through which the packet passes may exercise some degree of error control.

Figure 1. shows a typical composition of a packet of information bits. This particular example is taken from a packet radio network. [16]

In a packet-switching network, the packet represents the fundamental unit of transportation. One message may be broken into several packets and each packet may be independently routed to its final destination. Of course, at the destination the packets must be re-assembled in the correct order to reconstruct the original message.

Because packets of the same message can be sent by different routes, congestion on the network can be decreased. Fach packet contains its own control information in the header, and there are no lengthy connection and disconnect times as in the case of circuit switched systems.



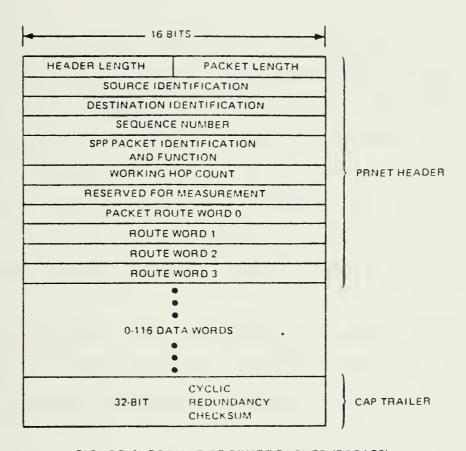


FIGURE 1 FORMAT OF PRNET PACKET (PRCAP3)



It should be noted that messages which are broken into packets are only meaningful within the network. When packets are passed through gateways into other packet-switched networks, a new intra-net level of protocol is required.

### C. THE ARPANET

Perhaps the best known example in the military of a large scale distributed system is the ARPANET. This research effort has been sponsored by the Defense Department's Advanced Research Projects Agency (ARPA). The ARPANET is a non-secure, packet-switched, distributed computer communication network which links together the computing facilities at universities and military installations across the continental United States and reaches overseas to London and Hawaii.

The justification and advantages of a network such as the ARPANET are summarized in the following excerpts from the text Computer-Communication Networks written in 1973:

[1]

"Che of the most successful aspects of the experiments in the use of time-shared computer systems conducted during the past decade was the ability to share computing resources among all the users of the system. Controlled sharing of data and software, as well as the sharing of the time-sharing system hardware, has led to much higher programming productivity and better overall utilization of the computing and user resources."



Even in these time-sharing systems, however, the system capacity was simply not large enough to perform all the storage requirements and computing potential that a decision maker required. There was the lack of a large enough community (critical mass phenomenon) in a single application area. Although it is possible to physically transfer programs or data from one community to another, this causes restrictions in language standards and hardware systems.

To quote further: "A viable alternative to program transferability, while permitting full resource sharing, is to provide a communications system that will permit users to access remote programs or data as if they were local users to that system. In addition, it should be possible for a user to create a program on his local machine that could make use of existing programs in the network as if they were available on his local machine. Pather than trying to move the programs from machine to machine, the network would allow the user or his program to communicate with a machine on which the program already executes. If enough machines can be connected into such a network, the total community in any particular application area would be sufficiently large enough to reach critical mass." [1]

This, essentially, is the rationale for military applications of the ARPANET research, both on the strategic and tactical level.



### D. MILITARY APPLICATIONS

We have spoken in a previous chapter of the tremendous growth of automated data systems. Tactical data systems, be they for the purpose of intelligence, pescripel management, fire direction, logistics, or command and control, tend to be engineered and developed separately without the consideration of total military mission. Nevertheless, the battlefield commander cannot make wise decisions based on input from just one of these systems. There is a good deal of interplay between all of these functional areas, and all must be considered.

One single tactical computer in a single command post could not be built to store and update all of the data represented in the combined systems. And if it could, such centralization would be unwise. The answer seems to be a distributed computer network built to interconnect and share the resources of the individual systems. This means overlaying an ARPANET-like architecture onto a series of distributed processors on the battlefield.

The AFPANET distributed architecture geographically separates data bases and computing resources. This distribution tends to decentralize a network, moving from a traditional hierarchical configuration to a grid or mesh-type configuration. Such decentralization is important to overall system survivability and reliability. A network architecture such as this is capable of remaining operational if one or more nodes is represent



non-operational. The architecture, combined with packet switching technology, allows for sufficient alternate routing capability to ensure a robust system. It offers significant improvement over some of the present "backbone" (hierarchical) systems in which the failure of one node along the chain would completely disrupt communications on the entire network.

It is obvious that a mobile and tactical application of ARPANET technology would offer new challenges to system engineering. Most apparent is the fact that ARPANET sites are interconnected by high speed, low-error, fixed telephone circuits. This kind of interconnectivity is not possible on a dynamic battlefield. The only other alternative is to utilize mobile, digital radio equipment to achieve connectivity.

# E. THE ALOHA SYSTEM

Several years ago researchers at the University of Hawaii began work on such hardware. Because there was an unusually high error rate on the local telephone lines, remote users of the university computer were unable to effectively communicate with the computing facility. This led to a research program to investigate the use of burst radio transmission in place of telephone lines for error-free, line-of-sight communications to the computer center. The resulting effort became known as the Aloha System, a series of packet-switched, ultra-high frequency (UHF), radio terminals. [13]



The Aloha System is essentially a broadcast, multi-access network. The "broadcast capability" of a radio channel implies that a signal generated by a radio transmitter may be received over a wide area by any number of receivers. "Multi-access capability" of a radio channel means that any number of users may transmit over a common channel. Hence, all users within line-of-sight of one another form a network that is completely connected, independent of the number of users.

## F. PACKET RADIO INTRODUCTION

Research done at the University of Hawaii led to the development of packet radio. Packet Radio extends the Aloha System to military uses. The goals of military experimental versions of a packet radio system differ from those of the original Aloha net in the following areas: [13]

- (1) Distributed control of the network management functions should be provided among multiple stations for reliability, and the use of a netted array of possibly redundant repeaters for area coverage as well as for reliability should be included.
- (2) The system should use spread-spectrum signaling for coexistance with other possibly different systems in the same band and for antijam protection. Surface accustic wave technology has become a viable current choice for matched filtering in the receiver.
- (3) The provision of authenticaion and anti-deception mechanisms is required.



- (4) System protocols should be incorporated that perform network mapping to locate and label repeaters, route determination and resource allocation, remote debugging, and other distributed network functions.
- (5) The use of various implementation techniques to provide efficient operational equipment such as repeater power shutdown except while processing packets should be included.

Because packet radios operate within a broadcast network, and all the network radios use a common frequency band (1710-1850 MFz), this technology has some favorable implications for frequency management and frequency conservation. According to current military doctrine, the frequency spectrum is allocated roughly in accordance with each user's stated requirement. In an Army or Marine Division this results in a frequency management problem of too many nets requiring too few frequencies and a constant threat of degraded communications due to mutual inteference problems. Once a frequency is allocated for a particular mission, it is not available for use by others in the same area.

This might be an effective management technique if each assigned band were actually used most of the time. In practice this is not usually the case, and much of the frequency spectrum is idle (not engaged in carrying traffic).



A broadcast network in which a number of users share a common broad frequency band offers improvement to this situation. The limited frequency spectrum could be used more efficiently if (1) the shared frequency band was wide enough to allow all users to transmit required traffic, (2) a channel access scheme was defined such that all users could access the channel when needed while at the same time allowing little or no mutual interference, and (3) the channel usage was high enough to ensure minimum empty time when the channel was not in use.

## G. CHANNEL ACCESS SCHEMES

One primary means of categorizing radio broadcast systems is the method employed for channel access. As mentioned earlier, packet communications have found important applications in ground-based radio information distribution, and in this situation there exists a common broadcast channel that is available and shared by a multiplicity of users. Because these users demand access to the channel at unpredictable times, some access scheme must be introduced to coordinate their use of the channel in a way which prevents degradations and mutual interference.

A large number of channel access ideas have been invented, analyzed and described in current literature. For a summary of these schemes, see [14]. All of these schemes, however, might be placed in one of three broad categories. [15] Fach category has its own advantages and costs.



# 1. Category I

The first category involves random access contention schemes whereby little or no control is exerted on the users in accessing the channel. This results in the occasional collision of packets on the air. A collision implies that at least one colliding packet is unintelligible and that channel usability for the time of the collision may be lost. Access schemes which fall into this first category are the pure Aloha, the slotted Aloha, and to a lesser extent, Carrier Serse Multiple Access. These are the access methods used by packet radio systems.

To better understand random access contention schemes, consider the example shown in Figure 2. There are some number of users, each of which transmits some number of packets of time duration "tau" at random times. Line four, labeled SUM, indicates the total traffic that all the users attempt to send in a fixed time.

In this example all traffic is able to be transmitted without conflict except for the collision indicated by the hashed area. These two packets (P-1 and K-1) may both be unrecognizable to the receiving station (at least one will be unreadable) and both could require retransmission.

In the non-slotted Aloha random access technique, packets are transmitted as soon as they reach the top of the transmit queue at the radio. No consideration is made of current channel activity. Therefore, they risk collision



with other packets on the air.

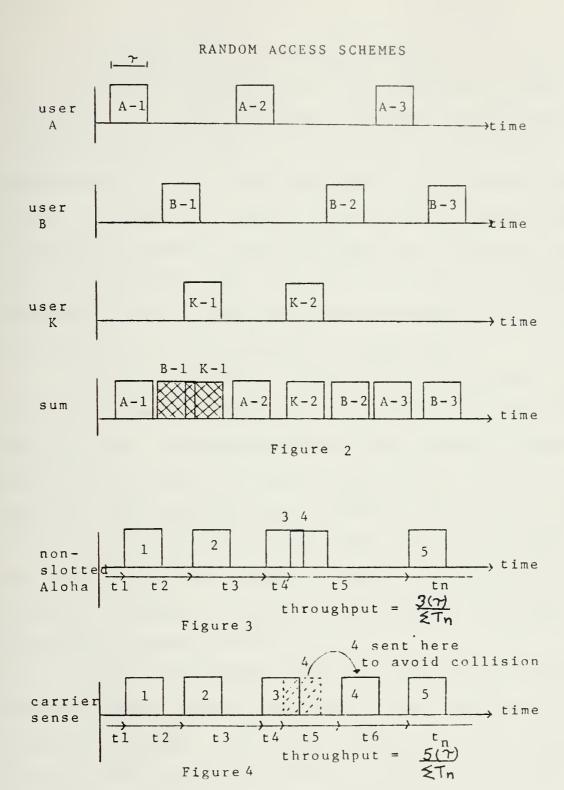
In the slotted Aloha method, time is broken into discrete quartities equal to the maximum time of propagation within the network. Users are restricted to transmission only at the beginning of each time slot. Again, collisions may be frequent.

In the carrier-sense mode, the radios listen before they talk and thereby reduce the risk of packet collisions. The radio senses the state of the channel before transmitting. If the channel is occupied, the radio waits a random amount of time and senses the state of the channel again before attempting to transmit.

An important measure of peformance for evaluating these random access techniques is "throughput." Throughput in packet radio technology is defined slightly differently than the definition given in Chapter II. Here it is defined as the percentage of time that the channel is actually occupied by useful traffic. Or, to put it another way, it is the message density on the channel. Is the channel carrying useful traffic (non-colliding packets) most of the time, or is there a lot of time wasted between packets when the channel is empty?

Figures 3 and 4 show how throughput is calculated in both techniques.







It should be obvious that throughput is higher in the carrier-sense mode due to the decreased number of collisions. In fact, analysis has shown that the maximum throughput possible with the non-slotted Aloha method is approximately 1/2e or 18.4%. Throughput in the slotted Aloha method is twice as high as the non-slotted method (36%). Throughputs as high as .80 to .90 have been obtained using the carrier-sense mode for cases in which the channel propagation time is small compared to message duration. Both of these methods are classified as asynchronous.

Another important measure of performance for distributed computer systems is a low delay time. This means that queries and responses between the system and the user take a minimum amount of waiting time. A short delay time is important when there are many interactive users on the net. Typically, in order to decrease delay time packet length is shortened. Long packet lengths, however, are necessary to increase throughput, shorten queue lengths and decrease processing overhead per bit.

On the battlefield, calls for fire from a forward observer would typically be short messages requiring a fast response, whereas, intelligence summaries are normally several pages long and require no reply. If both types of messages are carried by the same communications channel, one can readily understand why there are tradeoffs between throughput and delay time.



### 2. Category II

At the opposite extreme of categories of radio channel access methods, there are schemes which completely static reservation access methods. These schemes pre-assign capacity to users and effectively create as opposed to multi-access charnels. Such schemes as Time Division Multiple Access (TDMA), Frequency Division Multiple Access (FDMA), and Code Division Multiple Access (CDMA) fall into this category. The Joint Tactical Information Distribution System (JTIDS), Phase I, and the U.S. Marine Corps Position Location Reporting System (PLFS) the TIMA scheme in which time is broken down into use discrete intervals. The largest time cycle (called an "epoch" in JTIDS) is divided into thousands of smaller time slots or windows. Each user on the broadcast retwork is assigned one or many time slots in which he can transmit messages. After the passage of the cyclic time period his time slot again appears and he can transmit again. This technique, obviously, is highly dependent upon all users maintaining accurate time synchronization. The FDMA and CDMA schemes have also been developed to statically assign each user a fixed portion of the channel according to a unique frequency or code respectively. Here the problem is that a bursty user will often not use his preassigned capacity, in which case it is wasted. When a user is idle his portion of the channel cannot be used by other stations with traffic.



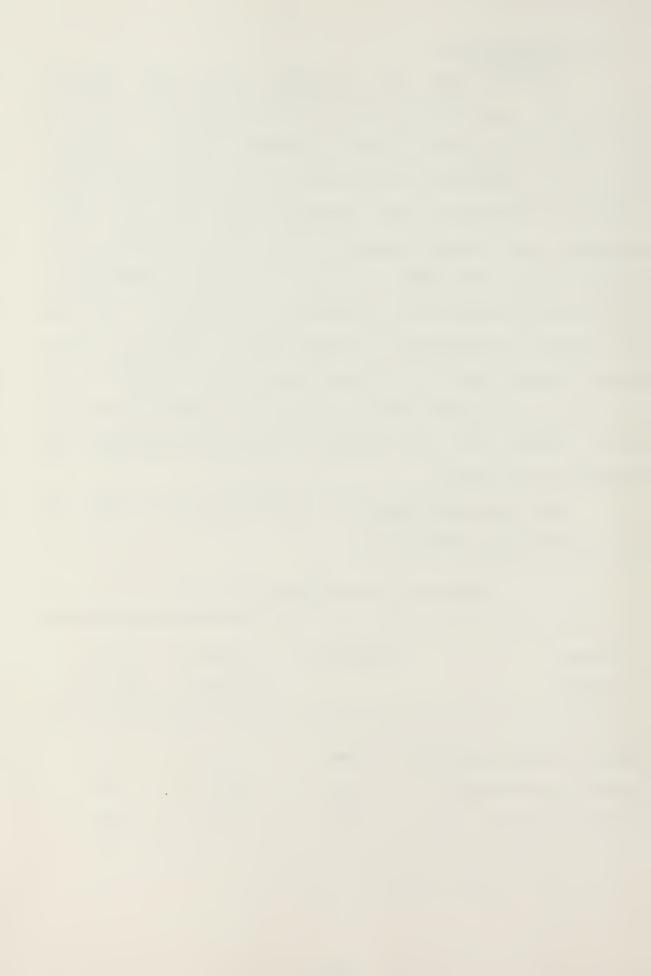
# 3. Category III

Between these two extremes are the dynamic reservation systems which only assign channel capacity to a user when he has data to send. In these schemes a certain portion of the channel is set aside in which to dynamically schedule transmission times. Several schemes fall into this category. In a Polling scheme the user waits passively to be asked if he has data to send. In an active reservation scheme the user asks for capacity when he needs it. In the Mini-Slotted Alternating Priority scheme a token is passed among numbered users in a prearranged sequence, giving each permission to transmit when he receives a token. The cost of these schemes is the overhead required to implement the dynamic reservations.

The following Figure 5. summarizes the costs of these three categories. [15]

The Cost of Distributed Resources

Access	Collisions	Control Overhead	Idle Capacity
Random Access Contention	yes	n o	r o
Dynamic Reservation	no	уes	no
Fixed Allocation	n o	no	ves



#### H. CURRENT PROGRAMS

As was previously mentioned, before large scale distributed data systems can be introduced to a mobile battlefield, higher capacity communications hardware is required. The U.S. Army is working on two programs to meet this requirement. The two systems under development are the PLRS/JTIDS hybrid and packet radio.

An Army Letter of Agreement [25] which addresses the need for these systems reads: "There is an urgent need for communications capable of supporting existing and programmed automated systems for Air Defense, Field Artillery, Intelligence, and Command and Control. Characteristics of machine-to-machine communications, coupled with the need for fast reaction times and a high degree of mobility, result in a requirement for a specialized distributed data communications system. Without this data communications improvement, highly sophisticated and highly effective weapon systems fielded in the early 1982's will not operate to full potential."

In further describing present communications systems this document speaks of existing equipment as being "technologically old, generally manually connected, too large and immobile (multichannel), and requiring intensive maintenance and logistical support." "The current communication system cannot, without the addition of a digital data communications capability, meet the demand imposed by the emergence of automated systems in the early



1980's."

The precise extent of this communications shortfall is a matter of intense study by the Integrated Tactical Communications Study (INTACS) Update Study effort.

The PLRS/JTIDS hybrid offers a solution to this problem in the mid-1980's. Packet radio, also, is a promising technology to satisfy future tactical data distribution missions. Its development schedule, however, effectively eliminates it as a short term candidate.

### 1. PLRS/JTIDS Hybrid

A detailed description of the PLRS/JTIDS hybrid is not possible here. Briefly stated, nowever, the system is a computer based system which provides real time, secure data communications, and position location and reporting information for tactical forces. It combines desirable features of both the PLRS an the JTIDS systems, using modified equipment from both systems. This system is planned for introduction to the field by 1986.

In addition to work being done on the PLRS/JTIDS hybrid, the Army is monitoring or participating in some interesting tests with packet radio, described in the following sections.

2. San Francisco Bay Experimental Packet Radio Network

The current experimental packet radio network being supervised by SRI International is located in the San Francisco Bay area and has been operational since July, 1976. Figure 6 [16] shows a map of the retwork sites.



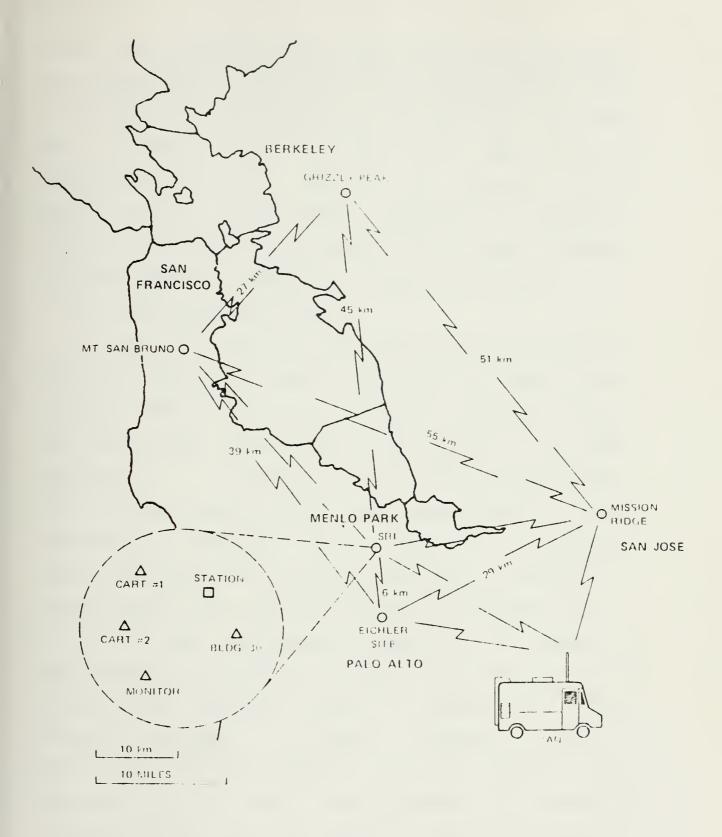


FIGURE 6 LOCATION OF MAJOR ELEMENTS OF THE PACKET RADIO TESTBED



There are two station PRU's located at the Menlo Park site. Each station has an associated PDP-11 computer attached. The network has four fixed repeater sites spread over the area and a variable number of packet radio user terminals (typically four to six). There are two vehicular packet radio terminals. These mobile terminals are an important aspect of the network, and a hand-off of a mobile terminal from one repeater to another is frequently exercised.

### 3. The Fort Bragg Test Bed

The U.S. Army has also recently set up a test bed for the evaluation of Packet Radio at Ft. Bragg, North Carolina. ARPA is sponsoring this effort, known as the Army Data Distribution System (ADDS). "The purpose of the ADDS experiment is to develop an environment in the resident XVIII Airborne Corps to permit user participation in the development, refinement and evaluation of innovative concepts for deployment of distributed data in support of future tactical Army data distribution requirements." [17]

In this multi-phased experiment, Ft. Bragg is experiencing a step-by-step build up of resources. The first phase of the experiment began in January, 1979. Three computer terminals, a network processor (called a TIP-Terminal Interface Processor) and a host computer were installed at Ft. Bragg and connected via commercial telephone lines into the ARPANET. After installation, operator training began in 1-2 day courses.



The second phase began in April, 1979 and the number of terminals was increased to fifty. The training in these phases acquainted operators of all ranks with the basic preprogrammed capabilities of the ARPANET including electronic mail, file management, directory maintenance, text editing, and printing. More specific applications geared to tactical information flow requirements are also being tested in garrison.

Phase III of the ADDS experiment is currently going on and involves the introduction of Packet Radio into the testbed. The PR network at Ft. Bragg will evertually grow to approximately 20 radios and 2 control stations. Initially, the radios are being employed in garrison to replace hard-wired connections. In the future, they will be deployed to the field in support of Corps exercises.

Packet Radic is expected to fullfill at least two major roles in the Ft. Bragg testbed. The first is to determine if this communications technology will satisfy the tactical data communications requirements of a corps on the battlefield. In this role, the system will be placed into the testbed as would any other communications system proposed for Army use. A second role is to provide a broad band communications channel for other systems under development which requires a data transfer capability. In this role the PRNET would provide communications for TACFIRE or some other intelligence or air defense system while maintaining its ARPANET connectivity as well. [17]



Reports from the Ft. Bragg testbed indicate that the XVIII Airborne Corps personnel have rapidly and enthusiastically adapted to the computer based communications technology. It is possible that the use of this data distribution technology can become as routine to commanders and staffs as voice communications are today.

[17]

In any case, the Ft. Bragg experiment represents an innovative and unique approach to investigating advanced concepts in Army doctrine and tactics. The testbed is a departure from traditional Army tests designed to arrive at production decisions, and is being driven by the urgent need for increased tactical data distribution capability.

#### I. FUTURE IMPLICATIONS

Is is not an exageration to say that these are very crucial days for the U.S. military. Resource investment decisions are more important now than ever before in history, and the consequences of bad judgement offer potential for great loss. Some would argue that an ever-increasing dependency upon technology by the armed forces is a very dangerous trend. Nevertheless, it is apparent that electronics and telecommunications advances are beginning to have significant impact upon strategy. Chapter One of the U.S. Army Combat Communications Field Manual is entitled "Command, Control, and Communications (C3)". A quotation from this document helps to stress the intensity of future battles and the necessity of accurate



and realistic planning for that engagement. [24]

"The U.S. Army has arrived at a point where technology and reality have outrun our old tactics on fighting and left ther in the dust. We have come to the shocking realization that the old way of doing things will not work any more."

"A good example of the change in combat reality facing today's soldier is an often used statistic from the Arab-Israeli War of 1973. In 20 days, over 1700 tarks were destroyed between the two sides. That's as many tanks as there are in five U.S. armored divisions. Technology has improved the weapons systems to the point where a tank has a 50-50 chance of being hit by the first round fired at it. We must retool our tactics to meet the reality of the next fight."

Certainly the capabilities of an army's command and control system has a great influence upon the capability of the force as a whole. The trends and technologies described in this chapter have far-reaching implications to doctrine and pose some problems that have yet to be solved. It is beyond the scope of this thesis to dwell on these implications in great length. A listing of the most obvious ones, however, is interesting and instructive.

# 1. Chain of Command

First of all, how will distributed systems change or affect the traditional chain of command structure? C3 system architectures typically reflect the chain of command within an organization, and this results in most C3 systems being



very hierarchical in structure. Distributed systems tend to be grid-like, peer structures. In a distributed system, is it advantageous to practice the concept of "skip echelon" reporting, in which certain levels of command may send traffic directly to the highest or lowest elements without intermediate "information only" stops? Certainly this would tend to decrease the time of delivery of messages and would avoid congestion on the network. But can the intermediate commanders afford to miss information that might prove to be critical or essential? What affect do distributed architectures have on the traditional role of communications centers and message centers? Would the requirement for these functions be eliminated? When the network is organized ir a peer structure, how are protocols designed in order to preserve a "priority" system to message traffic? These are questions that remain to be answered.

# 2. Network Management

Next, a very important question is the manner in which network control and data base management is exercised. Although distributed systems appear outwardly to be decentralized, a static, inflexible network control design would make the so-called "distributed" system as vulnerable as its hierarchical predecessor. Clearly the responsibility for network management and data base updating must be transferable within the network, and the capability should be shared by more than one station. There is a tradeoff reached, however, between one station and multi-station



operation. The more nodes there are which can exercise network control, the more vulnerable the entire network becomes to spoofing and deception techniques, and the more costly and complex it becomes. It is interesting to note also, that some of the hardest decisions and longest delays in the JTIDS program have dealt with this question of network management.

## 3. How Much Redundancy?

In addition to these important questions one must ask also, "How much redundancy is enough?" Although distributed networks are more survivable and redundant, what kind of back-up systems are still required? Certainly an increased dependency upon satellite communications in today's world has decreased the investment in long-haul, high frequency (HF) communications systems. Some have argued with convincing reasoning that this is a dangerous position. It is generally more attractive to engineers and planners to invent and employ new systems rather than to improve the old. Nevertheless, it is necessary to retain and maintain older, proven hardware as back-up, secondary equipment. The question remains, "Fow much is enough?"

# 4. Propagation Loss Due to Higher Frequency

Because of the congestion existing at lower frequencies and the high bandwidth requirements of new automated systems, new communications equipment is being designed to operate at very high carrier frequencies. This restricts propagation to strictly line-of-sight distances.



While this situation is acceptable in ground-to-air and ground-to-satellite communications, it poses serious constraints upon ground-to-ground mobile communications, especially in rugged or forested terrain. The amount of propagation loss in this situation is nightly significant, and the utility of these systems has yet to be proven.

## 5. Management Information Systems

It was mentioned previously that the development of an effective Management Information System (MIS) to integrate and display data to a commander and his staff is a formidable task. The Army, in fact, has been wrestling with this problem for over twenty years while attempting to field its Tactical Operations System (TOS, which is, in essence a MIS). After that amount of time, one questions whether such a system is nearer completion now than it was in 1960.

# 6. Interoperability

The military is fast finding out that interoperability means much more than mutually compatible equipment. It also means compatability of procedures, software, and message formats. One should watch closely the continued development of JTIDS and attempt to judge the success of joint service programs. The interoperability requirement tends to introduce system complexity in an attempt to make the system "all things to all people." When one moves to the problem of interoperability in the MATO environment, the question again becomes, "How interoperable can equipment be without becoming too costly, too bulky and



generally ineffective?"

### 7. Voice vs. Data Circuits

Another important question yet to be resolved is the proper trade-off of voice and data circuits. Again. JTIDS can be the case in point. Some services seem to be side-stepping procurement committments partly due to a lack of definition of system capability in this area. Although tactical commanders tend to prefer voice channels, voice channels require an enormously greater bandwidth allocation than data channels. If a commander is given the choice of having one voice channel or ten data channels into his command post, which will he choose? Which should he choose? What will the system offer?

## 8. System Cost

Finally, the question that will be asked most often is simply: "How much will the system cost?" Costs are divided into at least four categories. There are hardware and software costs, (it is common knowledge that the latter are now a greater consideration than the former) and there are initial procurement costs and life cycle costs.

Included in these costs is the manpower question. Is it realistic to think that as systems become more and more highly technical that the personnel who fix, operate and manage the systems will be better educated and more professionally competent?



### IV. AN INTRODUCTION TO PETRI-NETS

#### A. INTRODUCTION

The purpose of this chapter is to introduce the reader to the particular modeling tool which forms the basis for this research. The history of Petri-Nets is first discussed. Then an explanation of how Petri-Nets work is presented. Following some simple examples, the chapter concludes with a brief summary of the Strengths and weaknesses of this modeling tool.

#### B. HISTORY

The Petri-Net is named after its discoverer, Carl Adam Petri. These nets were developed in his early work in 1962 in Germany. They soon came to the attention of Anatol Holt who was then leading an Information Systems Theory Project for Applied Data Research, Inc. The work of this group eventually led to the theory of "systemics" [5] which dealt with the representation and subsequent analysis of systems and their behavior. At this point the modern formalism and notation of Petri-Nets was introduced. Holt also demonstrated the usefulness of the Petri-Net model in the representation of systems characterized by concurrent processes.

Perhaps the single largest source of research and literature regarding Petri-Nets has been Project MAC at the Massachusetts Institute of Technology. The Petri-Net model



was introduced to the researchers at Project MAC due to the association of Holt's group to J. Dennis' Computation Structures Group. This group has produced several PH.D thesis, together with many reports and technical memos dealing with Petri-Nets. In addition, MIT has sponsored two important conferences dealing with Petri-Nets. The first was the Project MAC Conference on Concurrent Systems and Parallel Computation held at Woods Hole in 1970. The second was the Conference on Petri-Nets and Related Methods, held at MIT in 1975.

This work, begun at MIT and continuing at other centers in the United States, until recently tended to concentrate on the formal or mathematical aspects of Petri-Nets. This work bears resemblance to the research in automata theory. It attempts to analyze systems by representing them as Petri-Nets, formally manipulating the representation in such a way as to derive information relating to the behavior of the modeled system. Because of the simplicity and power of Petri-Nets, they are excellent tools to use in the analysis of concurrent or asynchronous systems. They are finding their way into a number of diverse applications.

Petri, himself, is still actively researching, expanding his original theory. His extensions have led to a form of general systems theory called "net theory". Holt is continuing his research, concentrating on system representation and analysis of the formal representations.



# C. HOW PETRI-NETS WORK

Simply stated, a Petri-Net is a model. More specifically, it is an abstract, formal model that analyzes the flow of information in systems. [5,19] Petri-Nets also describe not only the information flow, but the controls and constraints of such flow. A Petri-Net graph models the static structure of a system in much the same manner as a flowchart models the structure of a computer program. As a modeling tool, Petri-Nets are especially useful in modeling systems that exhibit asynchronous and concurrent activity.

A Petri-Net consists of a collection of "events" and "conditions." In graphical notation, conditions are conventionally represented by circles and events are represented by bars. The Petri-Net is given structure and the capacity for interaction by connecting events and conditions with arrows.

An arrow from a condition to an event signifies an input condition to that event and implies that every occurence of the event terminates the "holding" of that condition. An arrow from an event to a condition signifies an output condition, and in this case, the occurence of the event commences the holding of the output condition. The graphic notation for a condition which holds is the marking of that condition by a "token".

The behavior of a system may be thought of as the occurence of events as time progresses. If all input conditions to an event hold, the event can occur. This



results in the holding of the event's output conditions.

Conditions may also be called "places" or "nodes" and events may also be referred to as "transitions." In the Petri-Net model of a system, directed arcs connect places to transitions and transitions to places.

The Petri-Net is first given a particular structure of places and transitions, and then it is "executed." Execution is governed by a "firing" protocol.

Simply stated, a transition may "fire" (symbolizing the occurence of an event) when all input conditions or places into that transition are marked with a token. When all inputs into a transition are marked, the transition is said to be "enabled". Figure 7 shows an "enabled" transition.

Execution of the Petri-Net involves the cyclic checking of all transitions once during each time interval. Each transition that is found to be enabled is fired, and tokens are moved from the input places of the enabled transition to the output places of that transition. This procedure continues for a set number of iterations. The flow of tokens in the Petri-Net thus symbolizes the flow of information or control in the modeled system.

By devising special methods for marking the number of tokens at the Petri-Nets nodes, the system status can be accurately and effectively recorded. The state of the system is reflected by an ordered set of mark status indicators which correspond to the nodes of the graph structure.



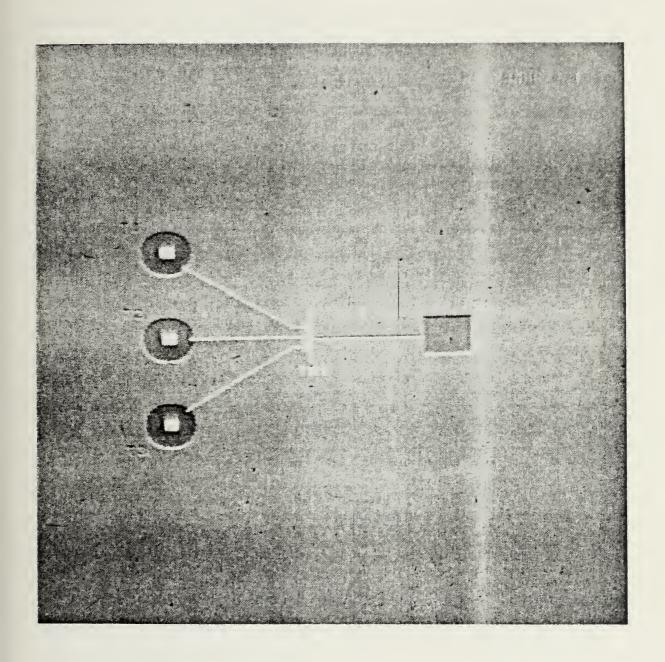


Figure 7.



In this thesis such an effective marking method is uniquely employed to give the viewer an accurate snapshot of network status.

Petri-Nets have rapidly gained acceptance over the last decade. Along with this aceptance has been the furthering of the understanding of Petri-Net properties.

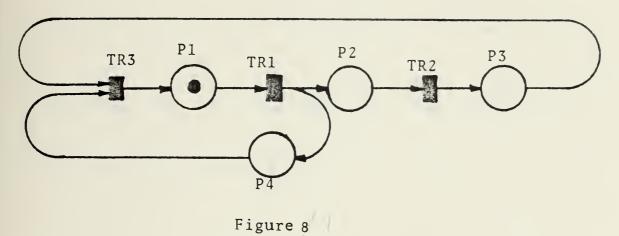
# D. A SIMPLE EXAMPLE

Figures 8 through 11 show the various states of a simple Petri-Net as execution occurs during four successive time intervals. [21] Notice four places (P1, P2, P3, and P4) and three transitions (TF1, TR2, and TR3). The directed arcs denote the interaction and relationships between input and output conditions. For instance, TR1 will become enabled when P1 (its only input condition) is marked with a token. At the time that TR1 fires, the token will be removed from its input condition (P1) and placed in its two output conditions (P2 and P4). In this manner flow of information or control is followed through the modeled system. Figure & shows the network at time = 0 with one token placed in P1. Figures 9 through 11 depict the Petri-Net as it continues execution through time = 3.

## E. ADVANTAGES AND DISADVANTAGES

The following characteristics of Petri-Nets were found by the authors to be strengths when using this particular modeling tool for simulations in the context of this research:





# A MARKED PETRI-NET (TIME = 1)

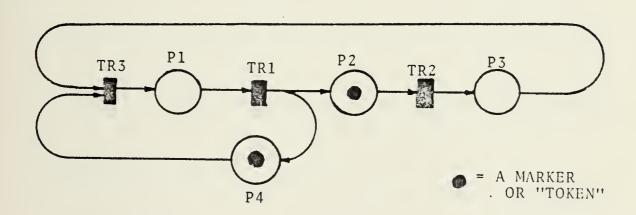


Figure 9



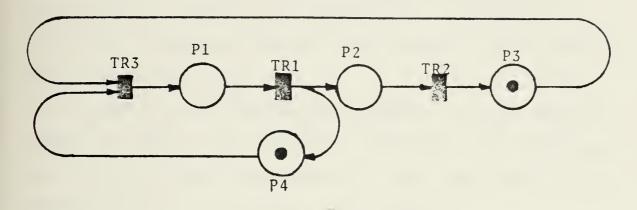


Figure 10

# A MARKED PETRI-NET (TIME = 3)

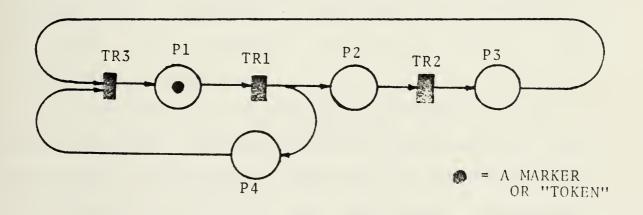
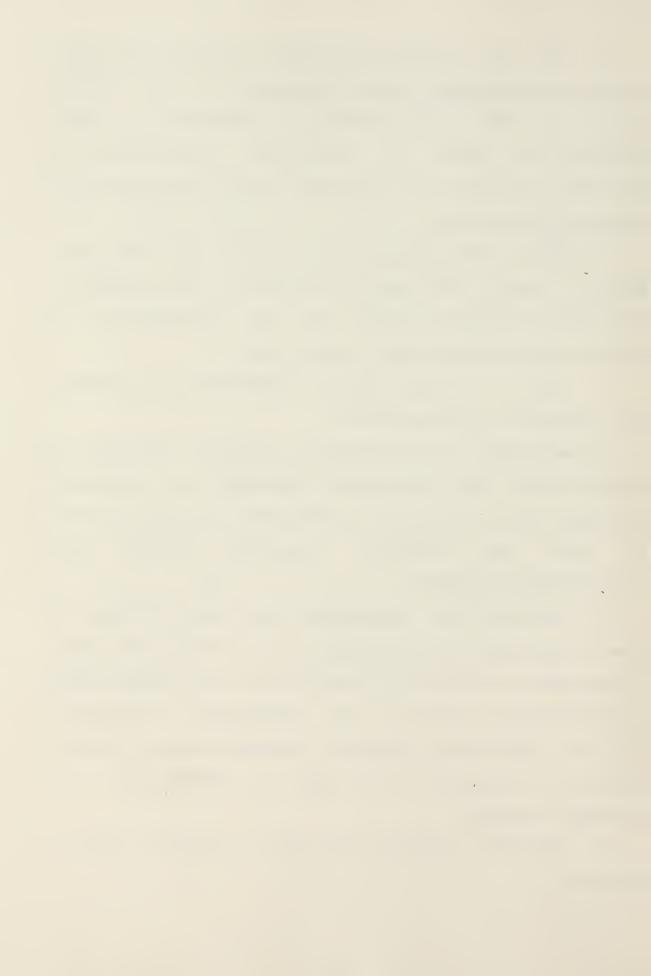


Figure 114



- 1. The rules governing Petri-Net execution are simple and easy to understand. This methodology can be quickly grasped by those with non-technical backgrounds who would ordinarily be unable to understand mathematical or analytical modeling. Yet Petri-Nets retain a night degree of precision and accuracy.
- 2. There is much flexibility inherent in the Petri-Net graph to model wide ranges of complexity. For instance, a model can be further abstracted by the replacement of a complex network of nodes by a single node.
- 3. There is a large degree of flexibility in assigning time intervals during execution.
- 4. Petri-Nets are well-suited to "snapshot" portrayal of network states. This advantage is important in simulation languages and is considered a strong point of languages such as GPSS and SIMSCRIPT. Petri-Nets possess this characteristic by nature.
- 5. Petri-Nets nave the potential for a wide variety of uses. Basically, any process that can be flow-charted could be expressed by Petri-Nets. Applications could include: flow of information or control in an organization, information flow in electronics hardware, representation of computer software or, procedures and stages of development in a management program.
- 6. Petri-Nets lend themselves well to computer graphics display.



7. Petri-Nets are very effective wher modeling concurrent, asynchronous activity in a network or system.

Certain weaknesses also became apparent to the authors in the course of this work. They are listed as follows:

- 1. Although Petri-Nets are basically simple to understand, the small building blocks of a network soon become exceedingly large and complex when large systems are modeled. The input files to some of the larger simulations in this paper were more than 1400 lines long. These networks must be drawn on paper before their entry into the computer, and this kind of effort soon becomes very tedious and prone to error.
- 2. Petri-Nets are best suited to concurrent, asynchronous behavior. When non-concurrent, synchronous behavior in a system is modeled the Petri-Net assumes a large amount of overhead.
- 3. The fact that Petri-Nets are not generally well known in the computer communications community could be a disadvantage when the user wishes to prove the accuracy of his rodel.
- 4. The fact that the simulator'employed in this thesis effort was deterministic could be considered either as an advantage or disadvantage depending upon the application. Many simulations have value because of their stochastic nature. Certain classes of experiments, however, need to be understood not because of "chance" happenings but because of the operation of the "laws" of nature working upon the



elements of the experiment.

As with any modeling technique, success is achieved through the modeler's familiarity with the modeling tool. Petri-Nets provide an excellent means to model those applications best characterized by their asynchronous, concurrent nature.



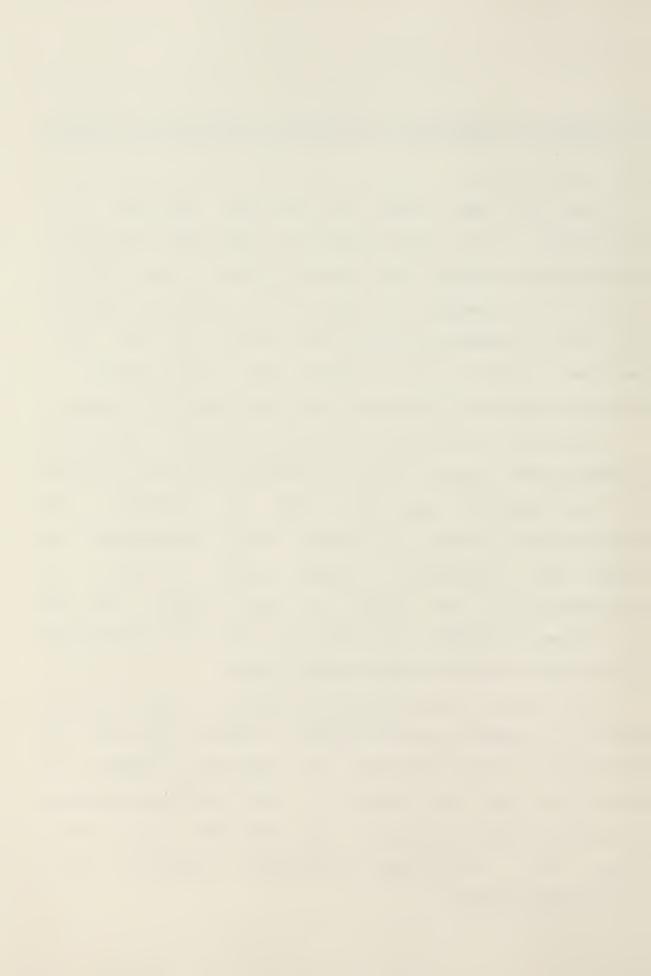
# V. PETRI-NET SIMULATIONS OF COMPUTER COMMUNICATION NETWORKS

### A. INTRODUCTION

Once the basic rules of Petri-Net execution are understood, it is a simple matter to apply these rules to a communications network. The system modeled is the network itself. The movement of tokens in the Petri-Net represent the flow of information within the system: in this case, message traffic in the network. Each token becomes a discrete amount of information contained within the message.

The places in the Petri-Net graph are used to represent communications nodes within the network. The directed arcs of the Petri-Net graph are used to represent the communications links or channels which interconnect the nodes. The transitions between nodes indicate the availability of the channel to carry traffic. If the transition is enabled, the channel is clear, and the message is relayed from input node to output node.

The careful structuring of the Petri-Net graph imposes upon the modeled system a variety of network protocols. An advantage of using Petri-Nets for simulation purposes is that the logic and protocol of the system are entirely contained in the structure of the Petri-Net graph rather than in a complicated mathematical algorithm within simulation software.



# B. TYPES OF NODES

The graphical output of the simulation attaches significance to the shape of the nodes displayed to the screen to facilitate recognition and interpretation. The experimental packet radio network in the San Francisco Fay area defines three primary types of nodes: terminals, stations, and repeaters. A terminal is a user node at which traffic is inputted or to which traffic is destined. It could be a fully automatic sensor, a handheld device, or a keyboard with CRT; but a terminal is a place where users connect to the network. The station is the node at which network control is exercised. The station typically keeps network statistics. Monitors flow and congestion control, assigns routing, and performs data base management for the system. It is a terminal with increased processing capability usually provided by an attached mini-computer. The repeaters are stand-alone devices placed in numerous, dispersed positions throughout the network to act as relay sites. Repeaters do not act as origins or destinations of traffic, but they serve the purpose of extending the geographical range of the network beyond a typical line-of-sight distance.

While all networks do not use this identical terminology, these three functional nodes summarize the requirements of communication hardware in most networks. In the simulations in this thesis, three types of figures and labels represent the functions of nodes as described above.



#### C. A SIMPLE APPLICATION EXAMPLE

In order to understand more completely the application of Petri-Nets to communications circuits, refer to Figure 12. In this diagram two distinct one-way communications channels are represented. The first goes from T1 to T2 and the second goes from T3 to T5 and through T4. The individual tokens are representative of packets of information in packet switched environment. The three additional nodes forming a triangle in the center of the diagram impose a special firing order upon the transitions in the communications channel. These center noies are systems overhead which ensure that only one "packet" can transmitted during a single time frame. In fact, if the three additional center nodes are thought of as a clock, then the entire network is a representation of Time Division Multiple Access in a network. A terminal can only transmit during a particular assigned time slot. After the time slot passes, the user must wait until the clock cycles back to his slot agair. Because each transmitter has a unique time slot assignment, no two terminals can transmit during the same time, and collisions of packets on the radio broadcast channel are eliminated.

This explanation should give the reader a simple idea of the manner in which various protocols are represented. Obviously the Petri-Net in Figure 12 allows only non-concurrent activity on the communications channels.



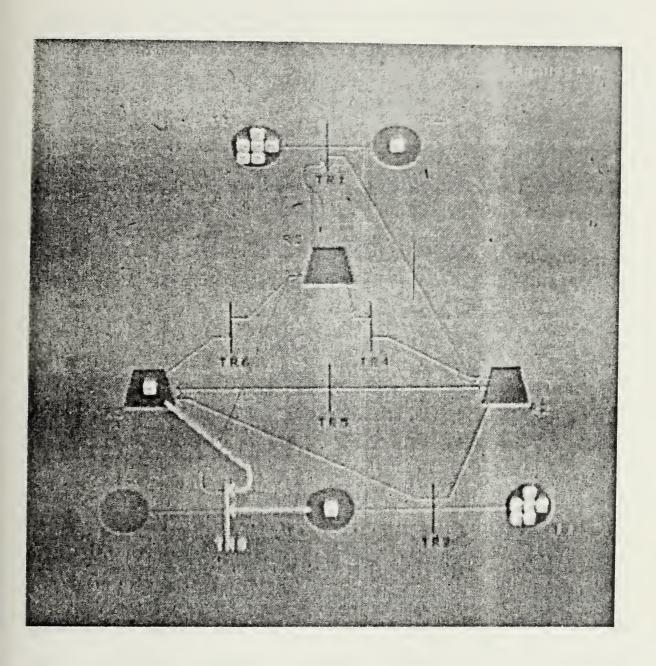


Figure 12.



The removal of the additional overhead would allow concurrent activity. In fact, this is sometimes desirable. For instance, those networks such as AUTODIN II which will make use of leased landlines will allow concurrent activity in the network. The Petri-Net is more efficient when modeling concurrent activity. The requirements to ensure non-concurrency as in Figure 12, causes one additional node and one additional transition to be placed in the Petri-Net graph for every transition on the communication links. Using the fundamental Petri-Net described here as a small building block, a system of considerable complexity can be built with many origin and destination terminals, and which allows packets to flow two directions with multiple, alternate paths from source to destination.

#### D. RANDOMNESS IN PETRI-NETS

The Petri-Net simulator used in this thesis work is deterministic. After the simulator begins execution there is no means to interactively alter the sequence of events, and there is no element of randomness within the simulator. Because of this situation, the same input file will always give identical output. Although the capacity for alternate routing within the communications network is implemented, the tokens do not randomly "choose" their routes during execution. They can only follow their pre-assigned routes from origin to destination. This means effectively that fixed routing instead of adaptive routing is represented in the simulations. This is not necessarily a disadvantage,



however, as some simulations have shown better throughput and time of delivery results using fixed routing ever adaptive routing [20].

It right be advantageous to modify the Petri-Net simulator to make it more stochastic in character. This might be accomplished in at least two ways. First, the inital marking of the network could be varied randomly at the beginning of each run. Certain key nodes could be marked or left unmarked according to the result of a call to some random number generator. Then the initial random state of the network would affect the end result of the output file.

A second way to add randomness deals with the Petri-Net concept of "dynamic conflicts" (see Figure 13, for an example of this particular network state). In this figure. the reader will notice that both Transitions 1 and 2 are enabled, but both cannot fire. Only one transition can fire, since in so doing it removes the token from T2 and disables the other transition. Thus TR1 and TR2 are said to be in conflict. This basic relationship can be used to create either deterministic or nondeterministic behavior. If the Petri-Net simulator is deterministic, the firing order for transitions in conflict is fixed according to a certain rule. This describes the case in this paper. The firing order of transitions is explicitly defined by their ordering in the input file. In the case of Figure 13, if TR1 is listed in the input file before TR2, and if a dynamic conflict occurs between them. TR1 will always fire first and



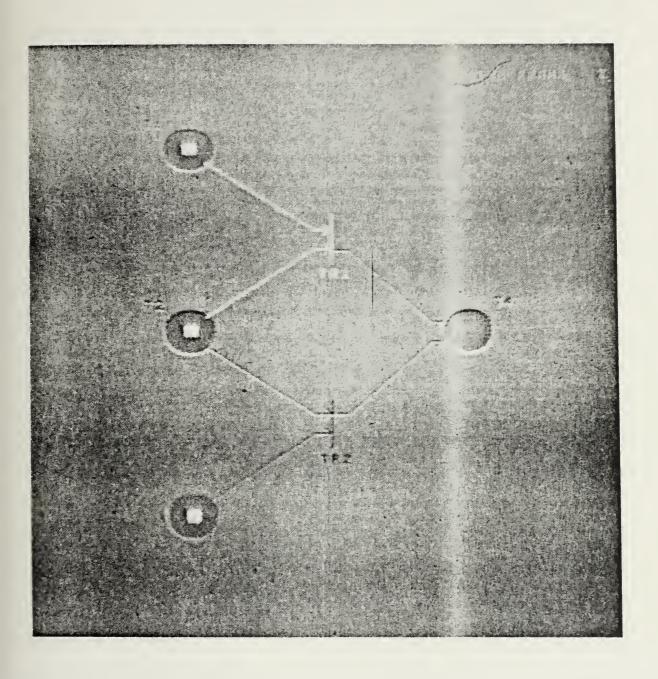


Figure 13.



TR2 will not fire during that time interval because it will have been disabled. This set of rules gives a strict priority of firing to the network.

The firing order of transitions in conflict could be modified so that it occurred in a random fashion. This would allow the Petri-Net to be executed in a non-deterministic manner and add the missing aspect of randomness to the outcome if so desired. In fact, Petri's first networks were non-determinestic because of this factor.

#### E. MEMORY STORAGE REPRESENTATION

The amount of memory storage in any particular communications node is also easily represented in the simulation. In this work a maximum number of seven packets is allowed at any one node. This number reflects the buffer size of seven packets in packet radio technology. If at any time a node accumulates more than seven packets, the buffer size has been exceeded and packets would theoretically be lost. When the buffer space is exceeded, the number of overflow packets is displayed outside the node in a red warning color.

#### F. SYSTEM LOAD AVERAGE

The system load on the network can be represented by the number of packets in transit at any one time interval. The system load then can be varied by controling the frequency of message generation. The shorter the interarrival time between generated messages, the busier the network will be.



At some point the network will be saturated and unacceptably congested if message input is greater than message throughput.

It is a simple matter to construct a message generator using Petri-Nets. See Figure 14. In this figure Transition 1 is firing to two outputs. One can be thought of as external and one output, R1, can be considered internal to the network. The external node can represent entry into the communications channel. The internal output feeds a token back into the "generator" and will, therefore, fire other packets into the network at regular intervals. The other repeaters (R2, R3) can be thought of as delays which slow down the frequency of message generation. This configuration of places and transitions constitute a message generator. The frequency of generation can be staggered and then several generators may be placed at the input of every communications circuit. In this menner the system load on the network may be varied.

#### G. TIME PEPRESENTATION

The Petri-net model is very good at representing the net status at distinct time periods. In fact, each tire interval displayed to the screen gives an excellent "snapshot" of network status. This is an important advantage inherent to the Petri-Net simulation. Another advantage is the flexibility afforded in assigning the time interval. The user has the prerogative of making each time interval as long or as short as is necessary. Successive snapshots may



represent the passage of time of 1 millisecond, 100 milliseconds, or 1 hour, depending upon the application and network being modeled.



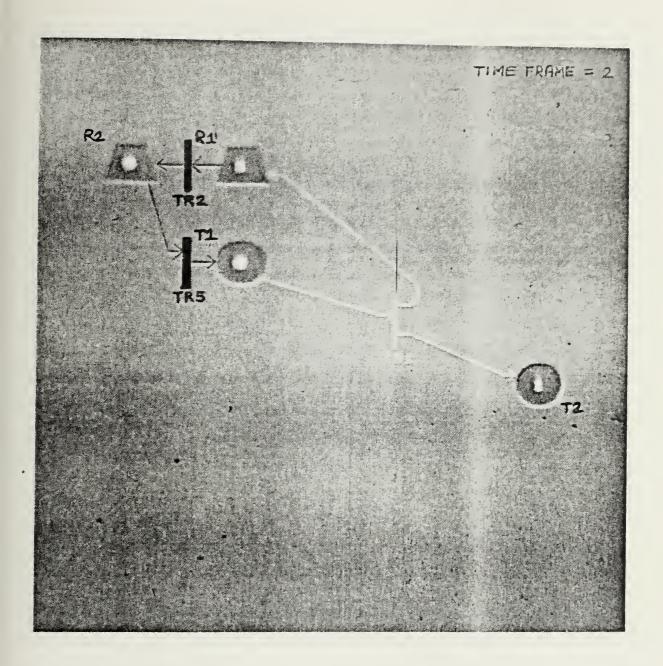


Figure 14.



## VI. THE EXPERIMENT

### A. DESCRIPTION

experiment was designed to demonstrate the feasibility of using the Petri-Net simulator as a predictor network performance. The experiment was performed by of keeping certain parameters constant and varying others. A series of six input files were run through the simulator. In six files the network architecture and the fixed routes were kept constant. Figure 15 shows the network. There were five origins of message traffic and five destinations. Four nodes were designated as terminals plus one station. There were four repeaters that performed relay functions within the network. Each of the five origins could send traffic to of four destinations. Each source to destination one combination had two routes for traffic to take. This made a total of 38 possible fixed routes in the network.

The controlled variables for the experiment were system load, concurrent vs. non-concurrent activity, and polling frequency of various circuits. The first three runs of the experiment were done with a high load. For the second three runs the message generators were slowed down to give a low system load. Some input files allowed concurrent network activity and some required non-concurrent activity. On two of the non-current runs the frequency of polling certain selected circuits was increased. This would represent the



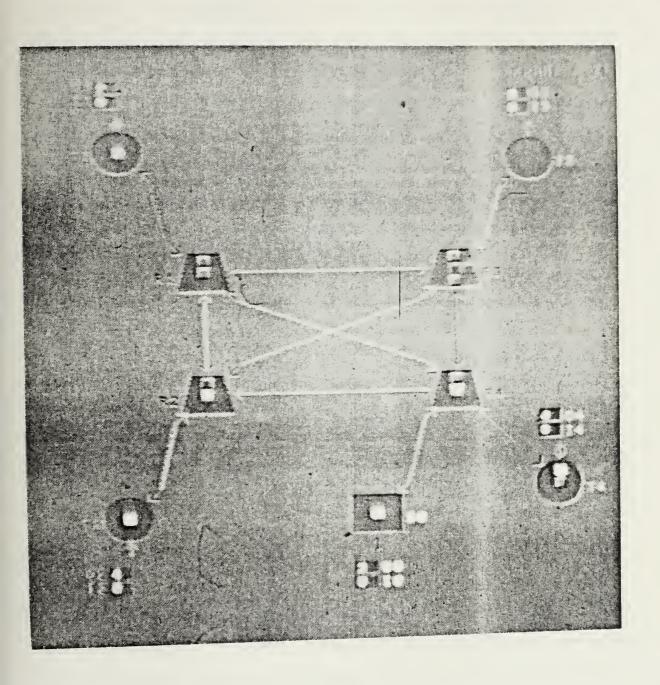


Figure 15.



equivalent of assigning a certain user more time slots in a TIMA scheme than another. It would give priority to those subscribers who have more traffic to send.

The only statistic gathered from the experiment was throughput measured in the number of packets which successfully reached their destination. This number could be extracted directly from the output queues at each terminal or station. The following points sumarize the characteristics of each of the six input files:

- 1. Runs 22-24 were run at high load.
- 2. Runs 25-27 were run at low load.
- 3. Runs 22 and 25 exhibit concurrent network activity.
- 4. Runs 23 and 26 exhibit non-concurrent network activity with equal polling frequency of all circuits.
- 5. Runs 24 and 27 exhibit non-concurrent network activity with weighted polling on certain circuits that are terminated at T1.

#### B. RESULTS

Figure 16 shows a summary of throughput statistics from the experiment. Certain results are no doubt obvious, but the quantitative nature of output statistics validates prior assumptions.

The following obervations are noteworthy;

1. Allowing concurrent activity on the network greatly increases throughput. The number of packets successfully transmitted in RUNS 22 and 25 was on the order of four times larger than the non-current runs. This question of



# THROUGHPUT IN NUMBER OF PACKETS

HIGH LOAD			
	RUN22	PUN23	RUN24
T1	92	16	25
T2	62	16	15
Т3	77	18	16
T4	58	18	17
S1	62	17	16
TOTAL	352	85	88
LOW LOAD			
	RUN25	RUN26	RUN27
T1	36	17	19
T2	38	17	16
T3	35	18	16
T4	33	16	15
S1	<u>31</u>	17	16
TOTAL	173	85	82

Figure 16.



concurrency has significant implications to radio broad-cast systems. For instance, what would be the advantages gained in the Packet Radio Metwork if terminal radio output power were reduced so that the terminal could only talk to its nearest neighbor rather than the entire network? This situation could allow simultaneous transmission of packets without the threat of collision. Also, a multiplexing scheme within the network could allow concurrent activity. These types of considerations could be modeled easily with the appropriate modifications to the Petri-Net graph.

- 2. Increasing the frequency of polling on selected circuits increases the throughput of those circuits. The reader should note the number of packets received at T1 in RUNS 23. 24. 26, and 27. RUNS 24 and 27 show a slight increase because certain circuits destined for terminal T1 were polled more frequently. This situation could easily reflect the assignment of more time slots to certain priority subscribers in a TDMA scheme. Again this modification was performed simply by restructuring the input file to the Petri-Net simulator.
- 3. The reader will note that the total throughput in the non-concurrent runs is largely the same regardless of the high load low load factor. This is because the system is basically "saturated" during non-concurrent activity at both high and low loading. Although the user might desire to improve throughput by generating more messages and trying to force them into the system, the network will give the same



results because it is already operating at full capacity.

4. A visual inspection of the output as PUN22 executes reveals that the system is essentially operating at peak efficiency under maximum load. The viewer will notice that the buffers at every location are frequently filled to capacity but seldom are overflowing.



## VII. RECCMMENDATIONS AND CONCLUSIONS

#### A. RECOMMENDATIONS

The potential exists for significant follow-on work to this thesis. Major topics for future work include the following:

- 1. The development of a language to describe networks, and the inclusion in the software of a "front-erd" program that would make the input file less cumbersome. The program could be static, like a compiler, or interactive, writter to query the user about a number of basic network parameters. For instance: How many nodes do you desire in the network? What are the paths of traffic from source to destination? Do you wish to allow concurrent or non-current activity? The software would take the user responses to such parameters and construct the input file from the responses.
- 2. A statistics gathering package should be written to collect and collate vital network parameters as the simulator is executed. Such a package would keep a running total of such items as:
  - a. average number of packets at a node
  - b. average number of packets on a circuit
  - c. percent use of a circuit
- d. average time delay between transmission and reception along each fixed route
  - e. number of messages lost



f. rumber of messages which successfully reach destination.

These statistics could be formatted and displayed by a post-processor.

- 3. As previously described, the addition of randomness to the simulator might be considered desirable, and , if so, the proper modifications could be added.
- 4. As the system approaches "production status", the entire question of model validation needs to be addressed carefully. Much could and indeed has been written on the subject of how to demonstrate your simulation is accurate.

  [12] As in most simulations there is a tradeoff between the degree of complexity represented in the model versus the largeness of the network. If a network is small or if only a small segment of the network is modeled, the degree of detail can be great. If the network is very large, however, the date storage capability might not allow the same level of detail.

There were two constraints to validation posed in this work. First, the memory capacity of the PIP-11/70 was stretched to the limit on the larger runs. The mini-computer offers 128,000 bytes of memory which are partitioned into thirds. This gives any one user 128/3 or 42.6 K bytes of memory. The graphics programs approached and then exceeded this bound before the work was completed. This forced the division of one program into two separate programs of 38K each. The file sizes for the output files from the Petri-Net



simulator are also very large. The larger of the input files produced output files on the order of 300K bytes. Figure 17. shows the relationship between number of nodes in the Petri-Net versus the size of the output files. In order to make claims of simulation accuracy based on real world networks, more memory is needed.

It should be added at this point that limitation of memory caused changes in the graphics display programs and in the overall organization of the software. The potential exists in the actual Petri-Net simulator (written in fortran and discussed in Appendix A) for the execution of Petri-Nets of well over a thousand places. If the user does not require a graphics output from this software package, then larger networks can be simulated on the PTP 11/70.

Secondly, the networks that were of most interest to the authors are largely experimental, unproven technologies. The Autodin II network is not yet operational and no statistics are available for validation purposes. The JTIIS technology is likewise not established operationally and many of the system characteristics are classified. Packet radio, which formed a good deal of background for the simulation, is still in its infancy. Also, packet radio employs a random channel access scheme which is contrary to the deterministic nature of the Petri-Net simulator.

Because these are new technologies, new routing and flow control algorithms and a host of different kinds of protocol are presently being developed. It is difficult to model a

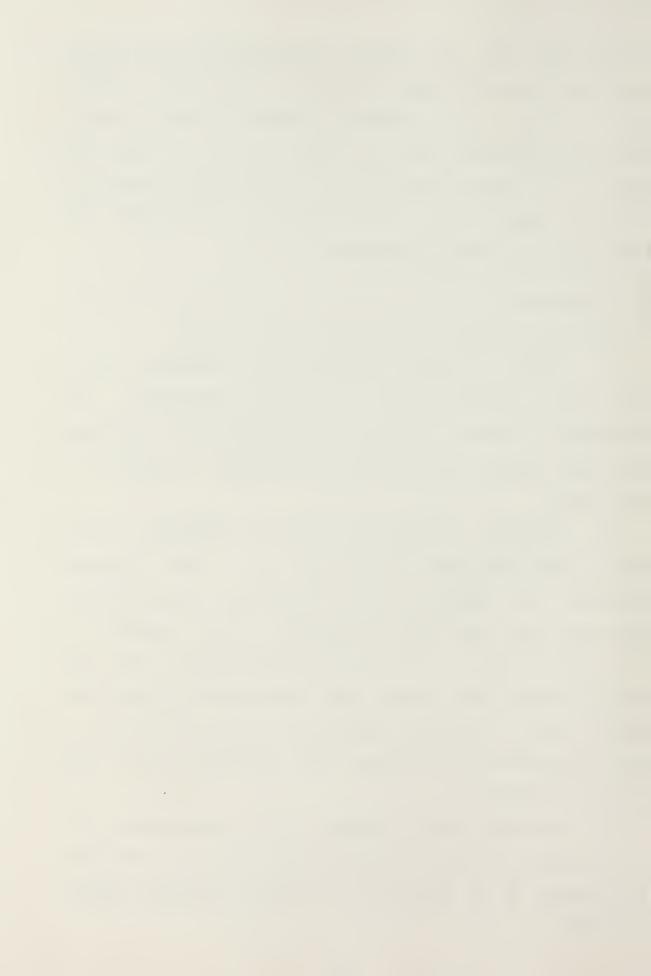


system that has not reached its production state. Pather than being concerned about modeling a particular network that is still in a process of charge and then trying to prove the simulation valid, it seemed wiser to leave the models in a more general state. By executing a variety of different input files, the simulations demonstrate the feasibility of future validation.

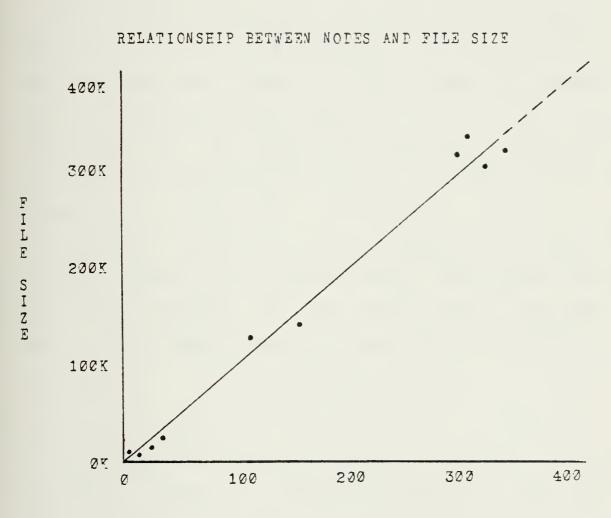
#### B. CONCLUSIONS

The conclusions of the authors are fourfold:

- 1. First, computer communication networks can be meaningfully modeled with the use of Petri-Nets. The background research to this thesis discovered no previous work which employs Petri-nets in the manner described in this paper.
- 2. Secondly, Petri-Net models of networks can be executed and displayed effectively on a color graphics terminal. The results of such a simulation are more easily understood than the common, hard-copy outputs produced by most analytical or queuing theory simulations. The color graphics output also could have considerable educational value. Again, background research uncovered ro instance in which Petri-Nets were displayed and executed on a color graphics terminal.
- 3. Thirdly, and perhaps most importantly, the implementation of such a modeling technique in a production environment as a predictor of system performance appears feasible.



4. Fourthly, there appears to be considerable benefits to encouraging future, carry-on work in the subject matter of this thesis.



NUMBER OF PETRI-NET NODES IN THE INPUT FILE

Figure 17.



## APPENDIX A - USER INSTRUCTIONS FOR THE PETRI-NET SOFTWARE

### A. INTRODUCTION

This chapter is written to describe certain procedures and syntax peculiar to the simulation software. Assuming that an interested student or faculty member is somewhat familiar with the theory and structure of Petri-Nets. the instructions in this section will allow him to apply the simulation and graphics output to his particular modeling problem.

Figure 18 shows the various components of the entire software package, the program source code sizes, the programming language, and the output files. The reader should refer to this figure as he reads the instructions in this appendix.

### B. THE INPUT FILE

The input file written by the user contains all of the information necessary to uniquely describe the Petri-Net model. This file is read by the fortran f4p program named. "simulator". The input file must be named "RUNYX, where XX can be any number from @1 through 99. There are three main divisions of the file,

Part I Places

Part II Transitions

Part III Marking



# SOFTWARE METHOLOLOGY

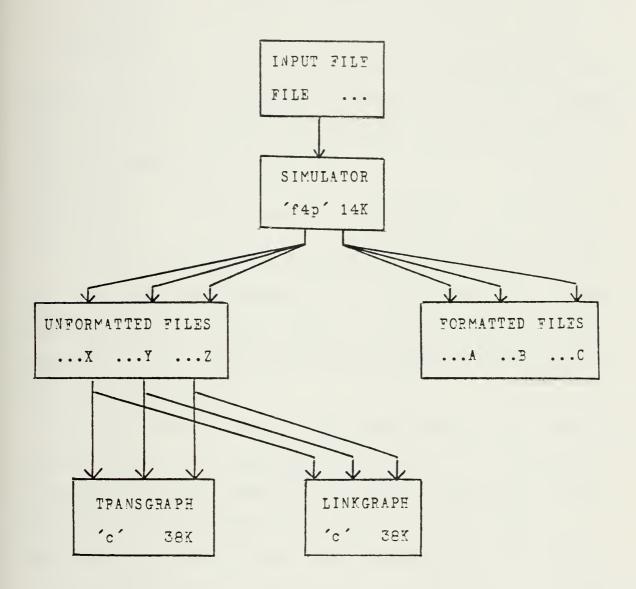


Figure 18.



Appendix B. shows a sample input file. The reader should refer to this example as he reads the following instructions.

## 1. Places

The first line of the file specifies the total number of places to be read in. Beginning on line number two, the names and locations of the places are listed. The following format applies:

Place Name .... X-cordinate .... Y-cordinate .... Plot Flag

The place name must be less than 13 characters long. Only the first two characters will be displayed as a label on the graphics terminal. There are unique instructions for labeling places in the multi-routing version. These will be discussed later.

The first letter of the name specifies the type of figure that will be displayed. The letter, "T", identifies a "terminal" and will appear as a circle on the output screen. The letter "S" identifies a "station" and will appear as a rectangle on the screen. The letter "R" identifies a "repeater" and will be displayed as a truncated triangle. Input and output queues can be represented by placing an "I" or "O" as the first character of the place name. These nodes are displayed as small rectangles, large enough to contain a two digit number. Names may begin with letters other than those listed above. They will not, however, be displayed to the screen.



After the name is listed, on the same line, the screen location of the place is specified by means of an x, y coordinate system. The x and y values are in the range of  $\emptyset$ -511, with the  $(\emptyset,\emptyset)$  point located at the top, left-hand corner of the graphics display unit.

The third item of information in the "places" line is a "plot on/off" entry. The user will frequently have places identified in the Petri-Net which are necessary control elements, but which do not need to be displayed on the output. A value of "1" will cause the place to be displayed with its label. A value of "0" is used for listing places which are not displayed to the output screen.

## 2. Transitions

After every place in the Petri-Net is listed (one line per place), the transitions are listed. As before, the first line specifies the total number of transitions to be read in.

Then, the transitions are each listed in a three-line format as follows:

Transition Name...X Coordinate...Y Coordinate...Plot Flag
Places into Transition
Places out of Transition

Transitions are named "TRXX", followed by their x.y coordinates, and a "1" or "3" to indicate whether they are to be displayed to the screen. The second line concerns the



input to the transition.

The first field of this line specifies the total number of inputs, and the following numbers indicate which places enter that transition. The numbers,

3 8 11 14

for instance, indicate that three places are inputs to this transition. The particular places are identified by their implicit, line-number ordering as entered in the list of places. In this example the three places are the 8th, 11th, and 14th places entered in the input file.

The third line of the transition entry concerns the outputs from that transition. The format is identical to that of the line above, i.e., the numbers

2 3 4

indicate that the transition fires to two outputs, the 3rd and 4th places listed in part one of the input file.

# 3. Initial Marking

After all the places and transitions are listed, the Petri-Net must be given its initial state of marking. The initial placement of tokens is specified by the following format:

MARK Line # of Place # of Tokens

Thus, the entry "MAPK 31" specifies that at the beginning of the simulation, Place 3 (the third entry in the list of places) is marked with one token. Several places may



be marked initially, but each marking requires a separate line.

Every line in the input file begins in the first column. Po not indent the beginning character of the line. The fields within each line must be separated by one (or more) blank spaces. The final line of the input file is the command "END".

## 4. Execution

When the user completes the input file he should exit from the edit mode and is now ready to execute the program "simulator" by typing "simulator.out". The program will ask him which input file he wishes to read, and the user responds by typing "RUNXX" (the file he previously created) and a <cr>

#### C. THE OUTPUT FILES

The program "simulator" produces six separate output files. When RUN01 is entered into "simulator", files named RUN01A, RUN01B, RUN01X, RUN01Y, and RUN01Z are produced. The files suffixed with A through C are formatted files. Files X through Z are unformatted. Files A and X contain the essential data structures that have been read in "simulator" from the input file. Files B and Y contain the markings for each place at successive time frames which will appear on the graphics output. Files C and Z contain information concerning which links or transitions fire at any particular time frame and are used to highlight present activity on the screen. The graphics display programs, written in "C",



require unformatted input files. The formatted files are necessary for the user to validate that correct input data is reaching the graphics program, and to troubleshoot when locating a problem. Examples of these files are contained in Appendices C through E.

#### D. USER OPTIONS

## 1. Choice of Programs

After the program "simulator" has executed the input file, the "C" programs read the output files and display the results of the simulation to the screen. At this point the user has several options concerning the method of display.

There are two separate programs the user can select—"transgraph" and "linkgraph". By executing "transgraph" the viewer will be able to observe the nodes of the network together with their associated transitions. "Linkgraph" does not display the transitions, but links node—to—node in the common way that communications networks are most frequently represented. For simple networks or to explain the basic working of Petri-Nets, the user will probably desire to see the transitions. "Transgraph" will only run with less than 100 places. For more intricate networks that contain several hundred places and transitions, the "linknode" program is necessary to avoid congestion on the screen.

# User Questions and Responses

After selecting which program to run, the user is given a series of questions from the CRT. Question one asks



the user to select which input file he wishes to execute. He responds with the command RUN X. For instance, if the input file named RUN21 has been executed by "simulator", and the user wishes to see the results, he enters RUN21X in response to question one. (Numerous files may be waiting in the users' directory which can be called in for display.)

Question 2 asks the user if he wants to view the data structures of the program. By entering a "1", the data structures will be printed to the CRT. A "2" will move on to the next question without viewing the data structures.

Cuestion 3 asks the user which of three versions of the program he wishes to see. Version 1 displays the marking of tokens in the conventional Petri-Net fashion, with numbers printed at the center of the places. Version 2 represents tokens by single, yellow boxes printed inside the nodes. These boxes are designed to represent "packets" of information in the packet switching concept. As previously described, each node has the capacity to hold seven packets. If the number of packets goes over seven, a red overflow number appears beside the saturated place.

The third version is the rulti-routing, multi-destination version. Version 3 uses color in a unique way. Because packets may be originating at different nodes and traveling to several destinations, the linking channels require two-way transmission. The graphics display is color-coded to highlight this information. Packets traveling to a particular place are colored to match the label of that



place. For instance, when a green box appears in the network, the viewer can trace its progress to the destination whose label is displayed in green. In Version 3 (multi-routing), a packet originating at a certain node and destined for another particular node may take different paths to arrive at the destination.

The user at this point must select one of the three versions by entering a "1", "2", or "3" followed by a <cr>.

Question 4 asks the user to select one of the three Genisco graphics terminals in the C3 lab on which the output will appear. Correct entries to this query are "0", "1", or "2".

The fifth question asks the user if he desires a time pause of two seconds duration between time frames of the simulation. If he desires the capability to look closely at each network snapshot, he enters a "1". If not, he should enter a "0" and the simulation will run without pauses. This gives the viewer more of a "real-time" impression.

At the end of this final question, the screen displays the initial condition of the network. After noting the initial condition, the user should type a carriage return to continue execution.

If the user has typed a "1" in response to question 5, he also has the ability to indefinitely suspend execution of the program at any time frame. He can interrupt the program by typing a "BRK" from the CRT keyboard. After studying that particular snapshot of network status, ne may



continue normal execution by typing a carriage return.

This interrupt does not work if the program is executing without pauses. Typing two consecutive "PRE's" will enable the user to exit the program entirely.

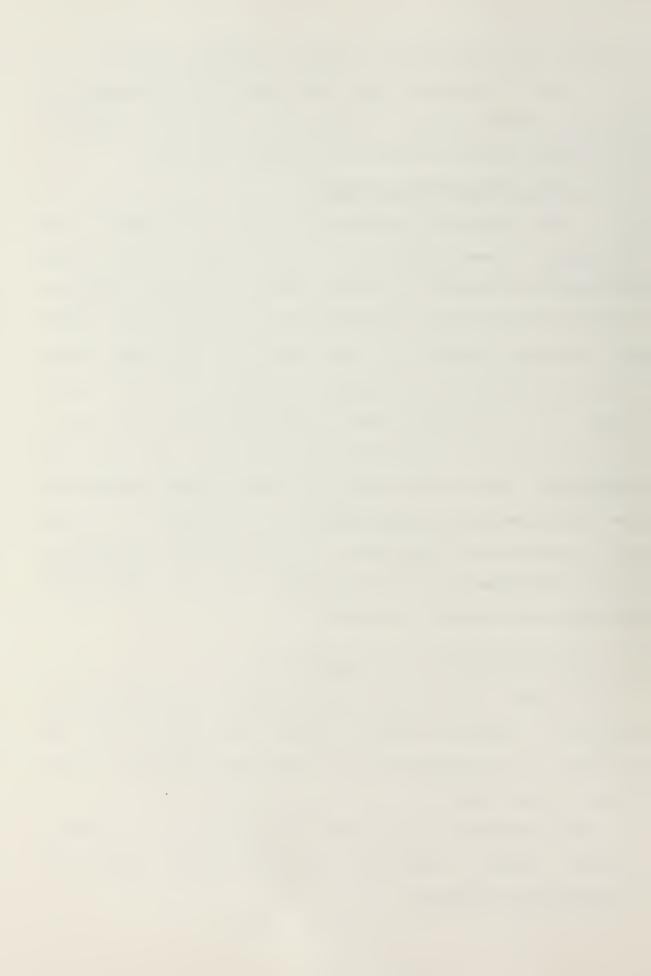
# 3. The "Highlighting" Feature

"linkgraph" draws lines in a dark blue color. The highlighting feature in both programs changes the blue cornecting link color to bright yellow on those links which are carrying traffic at any particular time frame. This feature performs in the following sequence. (1) The link lights up at the point where the future action will occur. (2) The packet(s) in question moves from one end of the highlighted link to the other. (3) The highlight remains on the link to emphasize where the action occurred. The user will notice that direction of movement on the highlighted link is indicated by an arrow pointing in the appropriate direction. See Figures 19 and 22.

### E. UNIQUE INSTRUCTIONS FOR VERSION 3:

In order for the user to implement the capabilities of Version 3, special network design information must be included in the input file. This paragraph describes these special instructions.

The simulation is structured to function in a fixed routing manner. That is, the multi-routed paths are predefined by the user.



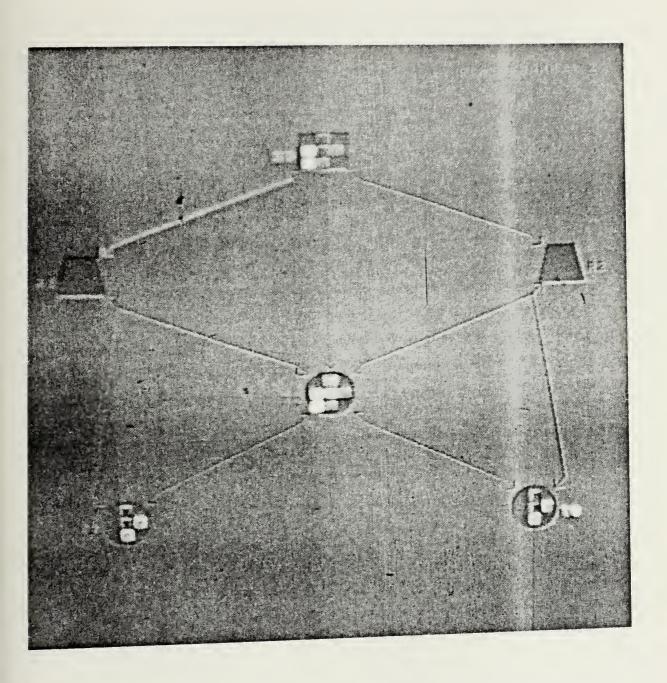


Figure 19.



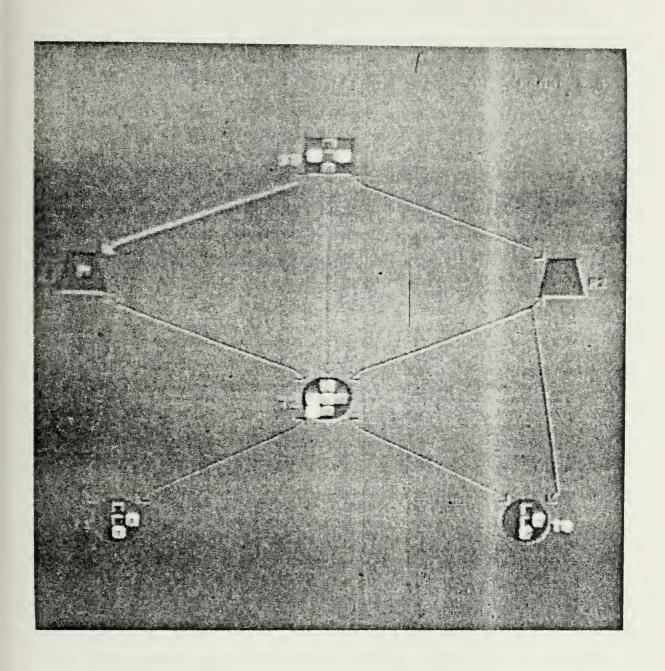


Figure 20.



There can be as many alternate paths from source to destination as the user wants to include. But once these paths are specified in the input file, they do not change dynamically during execution.

The multi-routing feature is made possible by "stacking" numerous places on top of one another and displaying these piggy-backed places at the same coordinates. In this manner the nodes appear on the screen as a single place, although in reality they may be buried several deep.

As the total network is conceptualized by the user, he must begin by mapping out all origins, all destinations and all relay nodes. Every node becomes unique to a particular path. For instance, a simple case would be to send a packet from T9 to T5 (see Figure 21.) by two different routes—Poute 1 goes through R1 and Route 2 goes through R2.

In this case T9 and T5 would be "stacked" two deep. The routes are T9-P1-T5 and T9-R2-T5. Because they are plotted at the same point, the terminals appear as a single node.

The "header" information to perform this routing is contained in the name of the place. In Version 3, every node that will be displayed to the screen must be assigned a seven unit name in the input file.

The first unit of the name specifies the type of figure to be displayed ("T", "S", "R", "I", "O" as previously explained). The second unit specifies the color that the name will be printed in. This number is derived from the particular color table ( 16 colors are in a color table )



that is being used by the graphics program. The color of the label must correspond to the color of the packets bound for that route's destination. Units 3 and 4 of the name designate the route number that the node lies upon. Route numbers are arbitarily given by the user and utilized for his own identification purposes. Unit 5 designates the color of the packet that will travel along that particular route. Every node on that route have the same color designator. The packet color of the route is determined by the destination of that route. Units 6 and 7 of the name specify how many nodes are stacked at that location. Places that are stacked must be listed together in the input file.

An example of this 7 unit name might appear as follows:

T205419

Field 1 designates that a circle will be drawn.

Field 2 specifies that the label will be displayed in color number 2 of the program's color table.

Fields 3 and 4 show the place on route number 05.

Field 5 ensures that every packet which passes through this place will be displayed in color number 4.

Fields 6 and 7 specify that places are stacked 19 deep at this coordinate.

As the program executes, the stacking algorithm totals all the packets which are located at a particular stacked location and displays that total number of boxes in the appropriate colors.



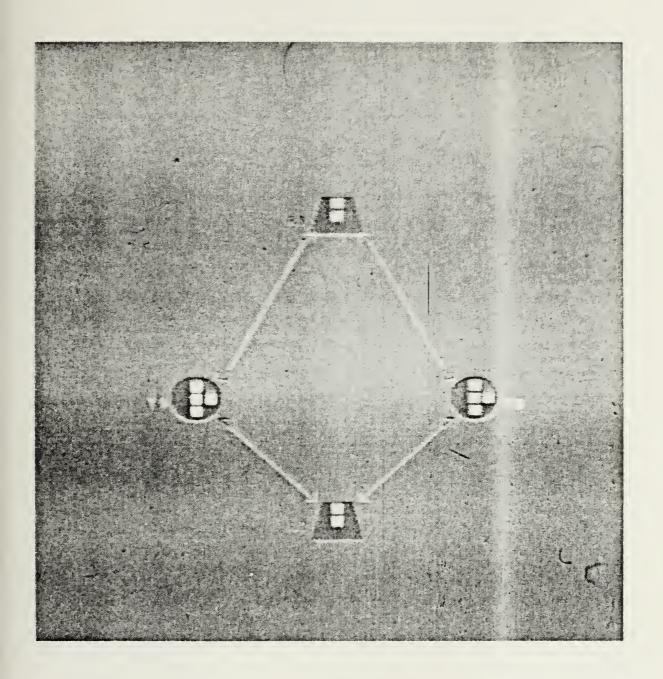


Figure 21.



```
Page 1 Fri Feb 8 05:09:47 1980
RUN03
  1 10
  2 T1 150 100 1
3 T2 150 250 1
  4 T3 150 400 1
  5 14 350 250 1
  6 G1 0 0 0
  7 G2 0 0 0
  8 G3 0 0 0
  9 G4 0 0 0
 10 G5 0 0 0
 11 G6 0 0 0
 12 8
13 TR1 250 175 1
14 2 1 2
15 1 4
 16 TR2 250 325 1
 17 2 2 3 -
 18 2 4 8
 19 TR3 0 0 0
 20 1 5
 21 2 1 6
 22 TR4 0 0 0
23 1 6
 24 1 10
 25 TRS 0 0 0
 26 1 7
 27 2 2 9
 28 TR6 0 0 0
 29 1 8
 30 1 3
 31 TR7 0 0 0
 32 1 9
33 1 7
34 TR8 0 0 0
  35 1 10
  36 1 5
  37 MARK 5 1
  38 MARK 7 1
  39 MARK 3 1
  40 END
```



RH 034		P 37	0 1		Tu 🕶 🕆	٦٢ .	1 4 1/2	5:37	:52	1 7 4 0				
1 **   F   V   T =   1   1														
,			1		U		150		1 (17)		1		5	
3			5		f <sup>n</sup> -		150		2511		1		6	
4			5		1		150		400		1			
5			7		0		350		250		1		ج	
5			Q		1		ij		0		0		2	
7			1.1		ō		0		Ü		Ú		2	
8			1.3		1		()		0		Ü		2	
3			15		0		0		0		0		2	
1.9			1.7		0		0		0		· 0		ج	
1.1			19		()		Ü		0		0		5 5 5 5 5	
12													_	
1.3	HXTT	PM=	3											
1.4		21		1		4		250		175		1		3
15		24		0		3		250		325		1		
16		27		12		1.4		0		0		()		3
1.7		30		1 7		10		0		0		U		3
1.8		33		21		23		0		0		J		3
10		30		25		2 14		υ		Ü		0		3
50		39		3 )		32		0		0		()		3 3 3 3 3
21		42		3.1		33		0		0		0		3
55														
23	IUTAR	LF:												
24	5	1	5	1	4	ج	2	3	2	4				
25	Я	1	5	S S	1	6	1	6	1	10				
25	1	7	5 2	2	0	1	2	1	3	1				
27	Q	1	7	1	1.0	1	5	0						
24														
53	MARE	3:	3, 4, 1	1 y 1 V 5	45									
3.0	11121	3T 4G	1620	3640	306 F	3110	:514	K T P AL	TR511	R61R7	TR8			
31														



1	1	1	1 0	0	1	0	0	1	0	
2	0	0	1 1	0	0	1	0	0	1	
3	0	1	1 1	1	0	0	0	1	0	
4	1	0	0 2	0	1	1	1	0	0	
5	1	1	1 2	0	0	0	0	1	1	
6	0	0	1 3	1	0	1	0	0	0	
7	1	1	1 3	0	1	0	0	1	0	
8	ō	ō	1 4	0	0	1	0	0	1	
9	0	1	1 4	1	0	ō	0	1	0	
10	1	0	0 5	0	1	1	1	0	Ō	
1.1	1	1	1 5	0	0	0	0	1	1	
12	0	0	1 6	1	O	1	0	0	0	
11 12 13	1	1	1 6	0	1	0	0	1	0	
14	0	Č	1 7	0	0	1	0	0	1	
15	0	1	1 7	1	0	0	0	1	0	
16	1	0	9 0	0	1	1	i	0	0	
17	1	1	1 8	0	0	0	0	1	1	
18	0	0	1 9	1	0	1	0	0	0	
19	1	1	1 9	0	1	0	0	1	0	
20	0	0	110	0	0	1	0	0	1	
21	0	1	110	1	0	0	0	1	0	
55	1	0	011	0	1	1	1	0	0	
23	1	1	111	0	0	0	0	1	1	
24	0	0	112	1	0	1	0	0	0	
25	1	1	112	0	1	0	0	1	0	
26	0	0	113	0	0	1	0	0	1	
27 28	0	1	113	1	0	0	0	1	0	
28	1	0	014	0	1	1	1	0	0	
29	1	1	114	0	Û	0	0	1	1	
30	0	0	115	1	0	1	0	0	0	
31	1	1	115	0	1	0	0	1	0	
32	0	0	116	0	0	1	0	0	1	
33	0	1	116	1	0	0	0	1	0	
34	1	0	017	0	1	1	1	0	0	
35	1	1	117	0	0	0	0	1	1	
36	0	0	118	1	0	1	0	0	0	
37	1	1	118	0	1	0	0	1	0	
38	0	0	119	0	0	1	0	0	1	
39	0	1	119	1	0	0	0	1	0	
40	1	0	020	0	1	1	1	0	.0	
41	1	1	120	0	0	0	0	1	1	
42	0	0	121	1	0	1	0	0	0	
43	1	1	121	0	1	0	0	1	0	
44 45	0	0	122	0	0	1	0	0	1	
	0	1	122	1	0	0	0	0		
46	1	0	023	0	1	1	1	1	0	
47 48	1	1	123 124	0	0	0	0	0	1	
49	0	0	124	0	1	0	0	1	0	
50	1	0	125	0	0	1	0	0	1	
20	0	U	163	V	U	Ţ	V	V	1	

RUN03B Page 1 Fri Feb 8 05:19:47 1980



```
Fri Feb 8 05:19:49 1980
RUN03C
      Page 1
 1 2
 2 3 5
3 3
4 1 4 7
```



```
RUN03C Page 2 Fri Feb 8 05:19:49 1980
 61 2
62 3 5
63 3
64 1 4 7
65 2
66 5 8
67 3
68 2 3 7
69 3
  70 4 5 6
 70 4 5 6
71 3
72 1 7 8
73 2
74 3 5
75 3
76 1 4 7
77 2
  78 5 8
79 3
 79 3
80 2 3 7
81 3
82 4 5 6
83 3
84 1 7 8
85 2
86 3 5
87 3
88 1 4 7
  89 2
90 5 8
 90 5 8
91 3
92 2 3 7
93 3
94 4 5 6
95 3
96 1 7 8
97 2
98 3 5
99 3
100 1 4 7
```



```
RUN20
         Page 1
                   Fri Feb 8 05:17:04 1980
  1 122
  2 $301212 250 100 1
  3 $302212 250 100 1
  4 $303112 250 100 1
  5 $304112 250 100 1
  6 $305012 250 100 1
  7 $306012 250 100 1
  8 $307312 250 100 1
  9 $308312 250 100 1
 10 $315312 250 100 1
 11 $316312 250 100 1
 12
   $319312 250
                100
 13 $320312 250 100 1
 14 R102205
            75 200 1
 15 R103105
            75 200 1
16 P110105
            75 200 1
 17 R114205
            75 200 1
 18 R115305
            75 200 1
 19 R205005 425 200 1
20 R208305 425 200 1
 21 R212005 425 200 1
22 R218205 425 200 1
23 R220305 425 200 1
24 1201216 250 300 1
25 1202216 250 300 1
26 T204116 250 300 1
27 1206016 250 300 1
28 1207316 250 300 1
29 1208316 250 300 1
 30 1209116 250
                300 1
 31 1210116 250 300 1
 32 T211016 250 300 1
33 7212016 250 300 1
34 1213215 250 300 1
35 1214216 250 300 1
36 T216316 250 300 1
37 1217216 250 300 1
 38 T218216 250 300 1
 39 T219316 250 300 1.
 40 1103108 100 400 1
41 1104108 100 400 1
42 T109108 100 400 1
43 7110108 100 400 1
44 T113208 100 400 1
 45 T114208 100 400 1
46 T115308 100 400 1
47 T116308 100 400 1
48 T005008 400 400 1
 49 T006008 400 400 1
 50 T011008 400 400 1
 51 1012008 400 400 1
52 1017208 400 400 1
53 T018208 400 400 1
54 T019308 400 400 1
55 1020308 400 400 1
            100 100 0
56 B1
            100 100 0
57 82
58 83
            100 100 0
 59 84
            100 100 0
            100 100 0
 60 B5
```



RUN20	Page 2	Fri	Feb 8	8 05:17:04	1980
8 UN 20 61 B6 62 B7 63 B8 64 B9 65 B10 66 B11 67 B12 68 B13 69 B14 70 B15 71 B16 72 B17 73 B18 74 B19 75 B20 76 B21 77 B23 79 B24 80 B25 81 B26 82 B27 83 B28 84 B29 85 630 86 B31 87 B323 89 B34 90 A1 91 A2	Page 2  100 100 100 100 100 100 100 100 100 1	Fri  100 0	Feb (	8 05:17:04	1980
97 A8 98 A9 99 A10 100 A11 101 A12 102 A13 103 A14 104 A15 105 A16 106 A17 107 A18 108 A19 109 A20 110 A21 111 A22 112 A23 113 A24 114 A25 115 A26 116 A27 117 A28 118 A29 119 A30 120 A31	500 500 500 500 500 500 500 500 500 500	500 0 500 0			



```
RUNZO
         Page 3 Fri Feb 8 05:17:04 1980
             500 500 0
121 A32
             500 500 0
122 A33
             500 500 0
123 A34
124 68
125 TR1
             220 200 1
126 2 1 56
127 1 23
128 TR2
             140 140 1
129 2 2 58
130 1 13
131 TR3
             140 260 1
132 2 13 60
133 1 24
134 TR4
             100 100 0
135 2 3 62
136 1 14
137 TR5
             100 100 0
138 2 14 64
139 1 39
140 TR6
             100 100 0
141 2 4 66
142 1 25
143 TR7
             100 100 0
144 2 25 68
145 1 40
146 TR8
             330 160 1
147 2 5 70
148 1 18
149 TR9
150 2 18 72
151 1 47
             375 320 1
152 TR10
             100 100 0
153 2 6 74
154 1 26
155 TR11
             100 100 0
156 2 26 76
157 1 48
             280 200 1
158 TR12
159 2 27 78
160 1 7
161 TR13
             365 260 1
162 2 28 80
163 1 19
164 TR14
             100 100 0
165 2 19 82
166 1 8
167 TR15
             175 400 1
168 2 29 84
169 1 41
170 TR16
             170 230 1
171 2 30 86
172 1 15
173 TR17
              50 300 1
174 2 15 88
175 1 42
176 TR18
             450 300 1
177 2 54 92
178 1 22
             360 140 1
179 TR19
180 2 22 90
```



```
RUN20
      Page 4 Fri Feb 8 05:17:04 1980
181 1 12
182 TR20 100 100 0
183 2 53 96
184 1 38
185 TR21 100 100 0
186 2 38 94
187 1 11
188 TR22
           100 100 0
189 2 52 100
190/1 21 /
191 TR23
            330 230 1
192 2 21 98
193 1 37
194 TR24 325 335 1
195 2 51 102
196 1 36
197 TR25 100 100 0
198 2 46 106
199 1 35
200 TR26 100 100 0
201 2 35 104
202 1 10
203 TR27 100 350 1
204 2 45 108
205 1 17
206 TR28 170 160 1
207 2 17 110
208 1 9
209 TR29 100 100 0
210 2 44 112
211 1 16
212 TR30 100 100 0
213 2 16 114
214 1 34
         180 310 1
215 TR31
216 2 43 116
217 1 33
218 TR32 10
219 2 32 120
          100 100 0
220 1 20
221 TR33 100 100 0
222 2 20 118
223 1 50
          300 400 1
224 TR34
225 2 31 122
226 1 49
227 TR35
            100 100 0
228 1 55
229 2 57 56
230 TR36
             100 100 0
231 1 57
232 2 59 58
233 TR37
             100 100 0
234 1 59
235 2 60 61
236 TR38
237 1 61
             100 100 0
238 2 63 62
239 TR39
           100 100 0
240 1 63
```



```
Page 5 Fri Feb 8 05:17:04 1980
241 2 65 64
242 TR40
            100 100 0
243 1 65
244 2 67 66
245 TR41
             100 100 0
246 1 67
247 2 69 68
248 TR42
             100 100 0
249 1 69
250 2 71 70
251 TR43
             100 100 0
252 1 71
253 2 73 72
254 TR44
            100 100 0
255 1 73
256 2 75 74
257 TR45
            100 100 0
258 1 75
259 2 77 76
             100 100 0
260 TR46
261 1 77
262 2 79 78
263 TR47
             100 100 0
264 1 79
265 2 81 80
266 TR48
             100 100 0
267 1 81
268 2 83 82
269 TR49
             100 100 0
270 1 83
271 2 85 84
             100 100 0
272 TR50
273 1 85
274 2 87 86
275 TR51
            100 100 0
276 1 87
277 2 89 88
278 TR52
            100 100 0
279 1 89
280 2 91 92
281 TR53
             100 100 0
282 1 91
283 2 93 90
284 TR54
             100 100 0
285 1 93
286 2 95 96
287 TR55
             100 100 0
288 1 95
289 2 97 94
290 TR56
             100 100 0
291 1 97
292 2 99 100
293 TR57
          100 100 0
294 1 99
295 2 101 98
296 TR58
           100 100 0
297 1 101
298 2 103 102
           100 100 0
299 TR59
300 1 103
```

RUN20



```
RUN20 Page 6
                 Fri Feb 8 05:17:04 1980
301 2 105 106
302 TR60
303 1 105
          100 100 0
304 2 107 104
305 TR61
           100 100 0
306 1 107
307 2 109 108
308 TR62
           100 100 0
309 1 109
310 2 111 110
          100 100 0
311 TR63
312 1 111
313 2 113 112
314 TR64
            100 100 0
315 1 113
316 2 115 114
317 TR65
            100 100 0
318 1 115
319 2 117 116
320 TR66
           100 100 0
321 1 117
322 2 119 120
323 TR67 100 100 0
324 1 119
325 2 121 118
326 TR68 100 100 0
327 1 121
328 2 55 122
329 MARK 1 1
330 MARK 2 1
331 MARK 3 1
332 MARK 4 1
333 MARK 5 1
334 MARK 6 1
335 MARK 27 1
336 MARK 28 1
337 MARK 29 1
338 MARK 30 1
339 MARK 31 1
340 MARK 32 1
341 MARK 43 1
342 MARK 44 1
343 MARK 45 1
344 MARK 46 1
345 MARK 51 1
346 MARK 52 1
347 MARK 53 1
348 MARK 54 1
349 MARK 61 1
350 END
```



```
DIEFCIUBA
                       - Lu- Mar 18 05:35:23 1980
             1 3 40 1
  2
  3
                THESIS-RELATED PROGRAMS & TEXT
  7
       FILE
                        DESCRIPTION
                                                     VERSION
                                                                 EXECUTION
  9
       DISECTORY
                       LIST OF THESIS FILES
 10
                                                                 aa DIRECTORY
                        GE, CBVIUS
                                                         2
       RUGALX
 1.1
                                                                 transgraph
       YCOMUSH
                        E AHLE THANSITION
                                                         5
 12
                                                                 transgraph
 1.3
       K11-103X
                        of AMIC CONFLICTS
                                                         2
                                                                 transgraph
       20%04X
                        HISPARCHICAL STRUCTURE HISPARCHICAL STRUCTURE
                                                         5
 1/4
                                                                 linkgraph
 15
       RUNOSY
                                                                 transgraph
       RU* Onx
                        RIG STRUCTURE
 16
                                                                 linkgraph
       2419:07Y
                        FERSHACK GELFRATUR
17
                                                         1
                                                                 transgraph
                        DESIGNS SYNCHRONIZATION
 14
       अंग्रिक्ष स
                                                                 transgraph
19
       RU-117
                        142 MULTIPLE INPUTS (5)
                                                                transgraph
20
                        TURNT 1 THEORIE
       PHATRY
                                                                transgraph
       RUL13X
                       COT SHIVER - PRODUCER
                                                         2
21
                                                                transgraph
                       CO SUME P-PRODUCER
55
       RUP14X
                                                         2
                                                                 transdraph
23
       KU...15X
                        THIME RUN PUROTX
                                                         2
                                                                 linkgraph
24
                        SYNCHROMIZATION II
                                                         2
       RUMIOX
                                                                 transgraph
25
                        EARKET-PADIO DETAORK
       FU\20X
                                                         3
                                                                 linkaraoh
25
       KII9/21X
                       ST PLE MULTI-ROHTING
                                                         3
                                                                 link/transgraph
27
       RU922X
                       CONCURRENT RETACRE
                                                         3
                                                                 linkaraph
28
                        II F-SECTIED COMMERCICATIONS
       4111123X
                                                                 linkgraph
29
       KU424X
                       PRIMPITY TIME-SCOTTED
                                                         3
                                                                 linkaraph
                        CO CHASENT REINDRY (-)
 30
       RHO25X
                                                         3
                                                                 linkgraph
 31
       2921 IIS
                        IT F-SLOTTED COVARMICATION(-)
                                                         3
                                                                 linkgraph
                        PRIMATTY TIVE-STOTIED (-)
 32
       PU/27.
                                                                 linkgraph
                        I PUT FILES TO SIVULATOR
 33
       RUM01-39
                                                                 simulator.out
                        THESIS ADVISORS
 34
       advisors.c
                                                                 advisors
35
                        EWE FRAME FOR FILM
       end.c
                                                                 end
       experiment.c
                        THEOUGHPUT RESULTS OF EXPRENT
 30
                                                                 experiment
                        INTERROCTION FOR FILM
 37
       intro.c
                                                                 intro
                        HORE-FILE SIZE RELATIONSHIP
 38
       linear.c
                                                                 linear
3.0
                        COTHUR GRAPHICS
       linkarach.c
                                                                 linkaraph
                        TITLE FRANK FOX 4PS
40
       nos.c
                                                                 りたら
                                                                 outline
 1.1
       outline.c
                        SCET, WARE MODULES
                        PETRIFIE STRUCTUR
 42
       simulator
                                                                 simulator.out
                       TITLE FRAME FOR FILM
43
       tirle.c
                                                                 title
                       TRA SITION GRAPPICS
34
       transprach.c
                                                                 transgraph
```



```
simulator
             Page 1
                        Fri Feb 8 07:10:27 1980
  1
          PROGRAM SIMULATOR
 5
   C
          ORIGINAL VERSION OF THIS PROGRAM (PROGRAM TESTNI) WRITTEN BY L.A.COXO
          MODIFIED TO OPERATE ON UNIX (PDP 11/70) BY S.C.JENNINGS & R.J.HARTELO
  4
   C
 5
          PPOGRAM SIMULATOR READS USER INPUT FILE AND PRODUCES 6 OUTPUT FILES
   С
 6
  7
   С
          MAINLINE
  8
 9
          CALL INIT
          CALL INPUT1
 10
          CALL DUMPPP
11
          CALL MOVENET (150)
12
          CALL EXIT
13
14
          END
15
16
17
          SUBROUTINE INIT
18
19 C
          INIT OPENS USEP INPUT FILE & CREATES 6 OUTPUT FILES
          FNAM1 STOPES THE IMPUT FILE ----- RUN..
50 C
          FNAM2 STORES THE FORMATTED INPUT DATA STRUCTURES ---- RUN..A
21 C
          FNAM3 STORES THE FORMATTED ITERATIONS OF THE NETWORK - RUN..8
55 C
23 C
          FNAM4 STORES THE FORMATTED LINKS OF THE NETWORK ----- RUN..C
24 C
          FNAM5 STORES THE UNFORMATTED GRAPHICS INPUT ----- PUN..X
25 C
          FNAM6 STORES THE UNFORMATTED GRAPHICS ITERATIONS ---- PUN..Y FNAM7 STORES THE UNFORMATTED GRAPHICS STATES ----- RUN..Z
26
27
28
29
          BYTE
                     FNAM1
30
          BYTE
                     FNAM2
                     FNAM3
31
          EYTE
32
          BYTE
                     FNAM4
33
                     FNAMS
          BYTE
34
                     FNAM6
          EYTE
35
          BYTE
                     FNAM7
36
          COMMON/USRFILE/FNAM1(6), FNAM2(7), FNAM3(7), FNAM4(7), FNAM5(7), FNAM6(7)
37
38
                          FNAM7(7)
          COMMON/EVENT/JEVENT(400,6), NXTEVT
39
          COMMON/TPANS/ITRANS(400,7), NXTTRN, IINTR, IISTOPE(100)
40
41
          BYTE NAMES
          COMMON/NAME/NAMES(4000), NXTNAM
42
          COMMON/IUTAB/IOTABL(4000), NXTTRE
43
44
     1000 FORMAT(' INITIALIZING PROGRAM')
45
          TYPE 1000
     1002 FORMAT(' BEGIN TEST-GRAPH-NET')
46
          TYPE 1002
47
     2007 TYPE 2000
48.
     2000 FORMAT (' ***-> INPUT FILE?
49
                                           NAME MUST BE ENTERED AS: RUN01 - RUN90
          ACCEPT 2001, FNAMI
50
51
     2001 FORMAT (6A1)
52
          FNAM1(6) = 0
          OPEN (UNIT = 1, NAME = FNAM1, TYPE = 'OLD', ERR = 2006)
53
54
          GC TO 2004
55
     5006
           TYPE 2002, FNAM1
     2002 FORMAT (' ERROP OPENING FILE ', X6A1)
56
57
          GO TO 2007
     2004 DO 2005 I = 1, 5
58
59
          FNAM2(I) = FNAM1(I)
60
          FNAM3(I) = FNAM1(I)
```



```
simulator
               Page 2
                          Fri Feb 8 07:10:27 1980
            FNAM4(I) = FNAM1(I)
  61
            FNAM5(I) = FNAM1(I)
  62
            FNAMb(I) = FNAM1(I)
  63
            FNAM7(I) = FNAM1(I)
  64
  65
      2005 CONTINUE
            FNAM2(6) = 'A'
  66
  67
            FNA^{M}2(7) = 0
            FNAM3(6) = 'B'
  68
  69
            FNAM3(7) = 0
  70
            FNAM4(6) = 101
  71
            FNAM4(7) = 0
  72
            FNAMS(6) = 'X'
  73
           FNAMS(7) =
  74
           FNAM6(6) = 'Y'
  75
           FNAM6(7) = 0
  76
            FNAM7(6) = 'Z'
  77
            FNAM7(7) = 0
  78
           NXTEVT=1
  79
           NXTTRN=1
  80
           NXTNAM=1
           NXTTRE=1
  81
      3000 FURMAT(' INITIALIZATION COMPLETE')
  82
            TYPE 3000
  83
  84
           RETURN
  85
           END
  86
  87
           SUBROUTINE DUMPPP
  88
  89
           OPENS FNAM2 & FNAM4
  90 €
  91
           BYTE
                      FNAM1
  92
           BYTE
                      FNAM2
  93
           BYTE
                      FNAM3
  94
           BYTE
                      FNAM4
  95
           BYTE
                      FNAMS
  96
           BYTE
                      FNAM6
  97
                      FNAM7
           BYTE
  98
           COMMON/USRFILE/FNAM1(6), FNAM2(7), FNAM3(7), FNAM4(7), FNAM5(7), FNAM6(7);
  99
                            FNAM7(7)
          1
           COMMON/EVENT/IFVENT(400,6), NXTEVT
 100
 101
           COMMON/TRANS/ITRANS(400,7), NXTTRN, IINTR, IISTORE(100)
           COMMON/IDTAB/IOTABL(4000), NXTTRE
 102
           BYTE NAMES
 103
 194
           COMMON/NAME/NAMES(4000), NXTNAM
 105
           OPEN(UNIT=1,NAME=FNAM2,TYPE='NEW',INITIALSIZE=40000)
 106
      1000 FORMAT(' NXTEVT=', 14)
107
      1001 FOPMAT(5x,618)
           WRITE(1,1000) NXTEVT
108
109
           DO 1500 I=1, NXTEVI-1
110
      1500 WRITE(1,1001) (IEVENT(I,J),J=1,6)
      2000 FORMAT(/, ' NXTTRN=', 14)
111
      2001 FORMAT(1x,718)
. 112
           HRITE(1,2000) NXITRN
113
114
           DO 2500 I=1,NXTTRN-1
115
           WRITE(1,2001) (1TRANS(I,J),J=1,7)
      2500 CONTINUE
116
117
118
      3000 FORMAT(/,' IOTABLE:',/,60(1014,/))
119
           WRITE(1,3000) (IOTABL(I), I=1, NXTTRE)
      4000 FORMAT(/, NAMES:
 120
                                 NXTNAM=1, [4)
```



```
Page 3
                       Fri Feb 8 07:10:27 1980
simulator
     4001 FORMAT(X, 100A1)
121
122
          WPITE(1,4000) NXTNAM
123
          WPITE(1,4001) (NAMES(I), I=1, NXTNAM)
     5000 FORMAT(1H1)
124
125
          WRITE(1,5000)
126
          CLOSE (UNIT=1, DISPOSE='SAVE')
127
128
          OPEN(UNIT=1, NAME=FNAM5, TYPE='NEW', INITIALSIZE=14000,
129
         1 FORM= 'UNFORMATTED')
130
          WRITE(1) (NXTEVT)
131
          DO 5001 I=1, NXTEVI-1
     5001 WRITE(1) (IEVENT(I,J),J=1,6)
132
133
          WRITE(1) (MXTTRN)
          DO 5002 I=1,NXTTRN-1
134
    5002 WRITE(1) (ITRANS(I,J),J=1,7)
135
          WRITE(1) (NXTTRE)
130
          WRITE(1) (IOTABL(I), I=1, NXTTRE)
137
138
          WRITE(1) (NXTNAM)
139
          WRITE(1) (NAMES(I), I=1, NXTNAM)
140
          CLOSE (UNIT=1, DISPOSE='SAVE')
141
          RETURN
142
          END
143
144
          SUBROUTINE INPUT1
145
146
147
          COMMON/SCAN/IAORD(15,10), NUMBER
148 8000 FORMAT('
                       INPUT BEGINS')
    8001 FORMAT(
                        INPUT COMPLETE')
149
150
          TYPE 8000
151
          1 = 0
          CALL SCANR
152
153
          CALL XINIGP(1,1)
          DO 1000 J=1, I
154
          CALL INPUTE
155
156
    1000 CONTINUE
157
          CALL INPUTT
158
159
160
     2000 CONTINUE
          CALL SCANR
161
          IF ("ATCHS(1, 'END', 3).ER.1) GO TO 3000
162
          IF (MATCHS (1, 'MARK', 4).EQ.1) CALL MARKER
163
          GO TO 2000
164
165
    3000 TYPE 8001
166
167
          CLOSE (UNIT=1, DISPOSE='SAVE')
          RETURN
168
169
          END
170
171
          SUBROUTINE INPUTE
172
173
          COMMON/EVENT/IEVENT(400,6),NXTEVT
174
175
          BYTE IWORD
          COMMON/SCAN/INOPD(15,10), NUMB
176
177
          J=0
          READ A SINGLE EVENT LINE FROM INPUT AND
178 C
179 C
          STURE IT APPROPRIATELY
180
          CALL SCANR
```



```
simulator
              Page 4 Fri Feb 8 07:10:27 1980
181
          CALL STONAM(1, IEVENT(NXTEVT, 1), IEVENT(NXTEVT, 6))
182
183
           00 1000 I=2,4
           CALL XINTGR(I, J)
184
185
    1000 IEVENT(NXTEVT, I+1)=J
186
187
           NXTEVT=NXTEVT+1
188
           IF(NXTEVT.GT.400) GO TO 9000
189
           RETURN
190
    9000 TYPE 9900
191
    9900 FORMAT(' EVENT/PLACE TABLE OVERFLOW')
192
           CALL EXIT
193
          RETURN
194
          END
195
196
197
          SUBROUTINE INPUTT
198
199
          COMMON/TRANS/ITRANS(400,7), NXTTRN, IINTR, IISTORE(100)
200
          BYTE I YORD
201
          COMMON/SCAN/INORD(15,10), NUMB
202
203
          I = 0
204
          K = 0
          CALL SCANR
205
206
          CALL XINTGR(1,I)
207
208
          DO 1000 J=1, I
209
          CALL SCANR
210
          CALL STONAM(1, ITRANS(NXTTRN, 1), ITRANS(NXTTRN, 7))
211
212
         DO 2000 L=1,3
213
         LL=L+3
214
         CALL XINTGR(L+1,K)
215
    2000 ITRANS(NXTTPN,LL)=K
216
217
          CALL SCANR
          CALL STUIDT(NUMB, ITRANS(NXTTRN, 2))
218
219
          CALL SCANR
550
          CALL STOTOT (NUMB, ITRANS (NXTTRN, 3))
          NXITRN=NXITRN+1
221
555
          IF(NXTTRN.GT.400) GO TO 9000
    1000 CONTINUE
223
224
225
          RETURN
226
227
    3000 TYPE 9900
    9900 FORMAT(' TRANSITION TABLE OVERFLOW')
228
559
          CALL EXIT
230
          RETURN
231
          END
232
233
          SUBROUTINE STONAM(NWORD, NPOINT, KOUNT)
234
235
                 STORE STRING 'NWORD' FROM SCANNER INTO
236 C
                 NAME TABLE AND RETURN A POINTER 'NPOINT'
237 C
238
          BYTE IWORD, NAMES, BLANK
239
          COMMON/SCAN/INORD(15,10), NUMB
240
```



```
Page 5 Fri Feb 8 07:10:27 1980
simulator
241
          COMMON/NAME/NAMES(4000), NXTNAM
242
          DATA BLANK/1H /
243
          DO 1000 I=1,10
244
          KOUNT=I-1
245
          IF (IWORD (NAORD, I) . EQ. BLANK) GO TO 2000
246
    1000 CONTINUE
247 2000 CONTINUE
248
249
          IF (NXTNAM+KOUNT .GT.4000) GO TO 9000
250
251
          DO 3000 I=1, KOUNT
252
     3000 NAMES(NXTNAM+I-1)=IWORD(NWORD, I)
253
254
          NPOINT=NXTNAM
255
          NXTNAM=NXTNAM+KOUNT
256
          RETURN
257
     9000 TYPE 9900
258
     9900 FORMAT( ' NAME TABLE OVERFLOW')
259
260
          CALL EXIT
261
          RETURN
          END
262
263
264
265
          SUBROUTINE STOIOT (NUMBER, LINK)
266
          COMMON/IOTAB/IOTABL(4000), NXTTRE
267
268
          STORE IMPUTS AND OUTPUTS OF TRANSITIONS
269 C
270 C
          IN THE TABLE, RETURN THE LINK
271
272
          IF(NXTTRE+NUMBER .GT. 4000) GO TO 9000
273
274
          K=0
275
          DO 1000 I=1, NUMBER
276
          CALL XINTGR(I,K)
          J=(NXTTRE-1)+I
277
278 1000 IOTABL(J)=K
279
          LINK=NXTTRE '
280
          NXTTRE=NXTTRE+NUMBER
281
          RETURN
282
283
284
     9000 TYPE 9900
285 9900 FORMAT(' IO TABLE OVERFLOW (TRANSITIONS)')
          CALL EXIT
286
          RETURN
287
288
          END
289
290
291
          SUBROUTINE MARKER
292
293
          COMMON/EVENT/IEVENT(400,6), NXTEVT
294
          I = 0
295
          J=0
296
          CALL XINTGR(2,1)
297
          CALL XINTGR(3,J)
298
          IF(I.LT.1 .OR. I.GT.NXTEVT) RETURN
299
          IEVENT(I,2)=J
300
          RETURN
```



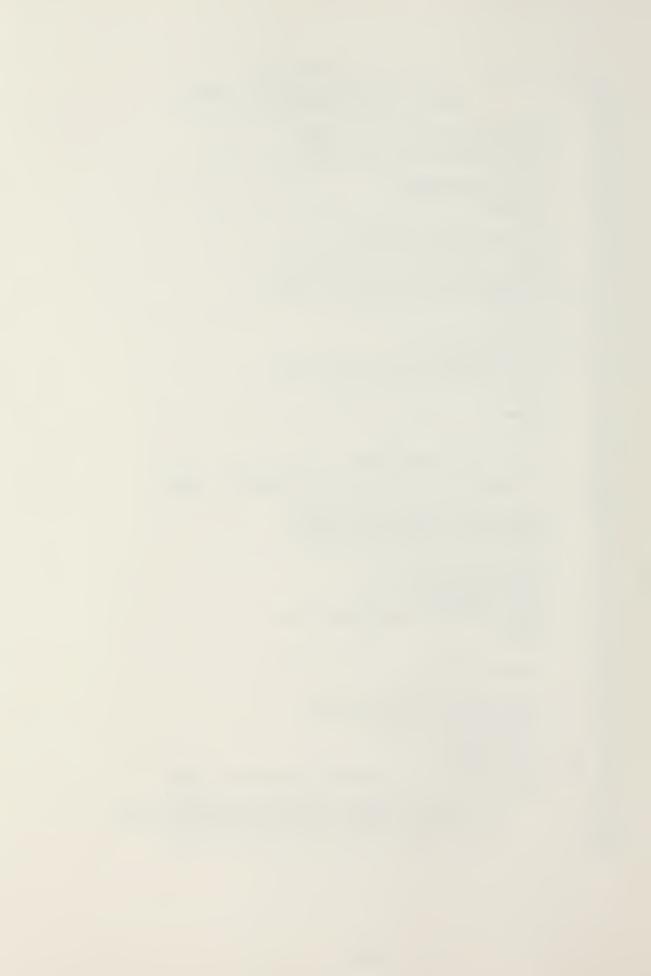
```
Fri Feb 8 07:10:27 1980
simulator
              Page 6
301
           END
302
303
304
           SUBROUTINE MOVENET (NTIMES)
305
306
307 C
           EXECUTE THE PETRI-NET 'NTIMES' OR STEPS
308
309
           COMMON/TRANS/ITRANS(400,7), NXTTRN, IINTR, IISTORE(100)
           COMMON/EVENT/IEVENT(400,6), NATEVT
310
           BYTE
                     FNAM1
311
                      FNAM2
312
           BYTE
313
           BYTE
                      FNAM3
           BYTE
                      FNAM4
314
                      ENAMS
315
           BYTE
           BYTE
                      FNAM6
316
           BYTE
                      FNAM7
317
           COMMON/USRFILE/FNAM1(6), FNAM2(7), FNAM3(7), FNAM4(7), FNAM5(7), FNAM6(7)
318
319
                           FNAM7(7)
320
          DATA ITIME/1/
321
322
     1000 FORMAT(' EXECUTING TIME=', 14)
323
           OPEN (UNIT=1, NAME=FNAM3, TYPE='NFW', INITIALSIZE=120000)
324
           OPEN (UNIT=2, NAME=FNAM6, TYPE='NEW', FORM='UNFORMATTED',
325
326
          1 INITIALSIZE=120000)
           OPEN (UNIT=3, NAME=FHAM4, TYPE='NEW', INITIALSIZE=12000)
327
          OPEN (UNIT=4, NAME=FNAM7, TYPE='NEW', FORM='UNFORMATTED',
328
329
          1 INITIALSIZE=12000)
330
331
332
          DO 2000 I=1, NTIMES
           TYPE 1000, ITIME
333
          IINTR=0
334
335
          CALL MOVE
      106 FORMAT (3512)
336
           WRITE(1,106) (IEVENT(J,2), J=1, NXTEVT+1)
337
           WRITE(2) (IEVENT(J,2),J=1,NXTEVT+1)
338
339
      107 FORMAT(I3)
340
          WPITE(3,107)
                          IINTR
341
      108 FORMAT(10013)
342
          WRITE(3,108) (IISTORE(J), J=1, IINTR)
           WRITE(4) IINTR
343
344
           ARITE(4) (IISTORE(J), J=1, IINTR)
345
346
           ITIME=ITIME+1
     2000 CONTINUE
347
348
349
          CLOSE (UNIT=4, DISPOSE='SAVE')
350
           CLOSE (UNIT=3, DISPOSE='SAVE')
           CLOSE(UNIT=2, DISPOSE='SAVE')
351
           CLOSE(UNIT=1, DISPOSE='SAVE')
352
353
354
           RETURN
355
           END
356
357
           SUBROUTINE MOVE
358
359
360 C
           EXECUTE THE NET ONE STEP
```



```
simulator
              Page 7
                        Fri Feb 8 07:10:27 1980
361
           COMMON/TRANS/ITRANS(400,7), NXTTRN, IINTR, IISTORE(100)
362
363
           DIMENSION MARKS (400)
364
           ITEST=0
365
           CHECK ALL TRANSITIONS TO SEE WHICH ARE ENABLED
366 C
367
           DO 0500 I=1, NXTTRN-1
368
     0500 MARKS(I)=NABLED(I)
369
           DO 1000 I=1, NXTTRN-1
370
           IF (MARKS(I).EQ.0) GO TO 1000
     0600 FORMAT(' DYNAMIC CONFLICT, TR#=',14)
371
372
373
           IF(NABLED(I).EQ.1) GO TO 0800
           TYPE 0600, I
374
           GO TO 1000
375
376
     0800 CONTINUE
           CALL UNMARK (I, ITEST)
377
378
           IF(ITEST.EQ.1) GO TO 1000
379
           IINTR=IINTR+1
           IISTOPE(IINTR)=I .
380
          CALL MARKEM(I)
381
382
     1000 CONTINUE
383
384
          RETURN
385
          END
386
387
388
          FUNCTION NABLED (NUMBER)
389
390 C
          RETURN 1 IF TRANSITION # 'NUMBER' IS ENABLED, READY
391 C
          TO FIRE. ELSE RETURN O.
392
393
          COMMON/TRANS/ITRANS(400,7), NXTTRN
394
          COMMON/EVENT/IEVENT(400,6), NXTEVT
395
          COMMON/IDTAB/IDTABL(4000), NXTTRE
396
397 C
          CHECK LIST OF INPUTS TO SEE IF ALL ARE MARKED
398
399
          MARK=0
          IPT=ITRANS (NUMBER, 2)
400
401
          KOUNT=IOTABL(IPT)
402
          DO 1000 I=IPT+1, IPT+KOUNT
403
404
          NEVENT=IOTABL(I)
405
          IF (IEVENT (NEVENT, 2).GT.0) MARK=MARK+1
406
    1000 CONTINUE
407
408
          NABLED=0
409
          IF (KOUNT.EQ.MARK) NABLED=1
410
          PETURN
411
          END
412
413
414
          SUBPOUTINE UNMARK (NUMBER, IERROR)
415 C
                UNMARK (IE. DECREMENT THE NUMBER OF TOKENS
416 C
                     ALL OF THE INPUT EVENTS TO TRANSITION # 'NUMBER'.
417 C
418 C
                     RETURN IERROP=0 .....UNLESS....
                     ONE EVENT IS A MULTIPLE INPUT OF THE SAME
419 C
                     TRANSITION AND WE DON'T HAVE ENOUGH MARKERS.
420 C
```



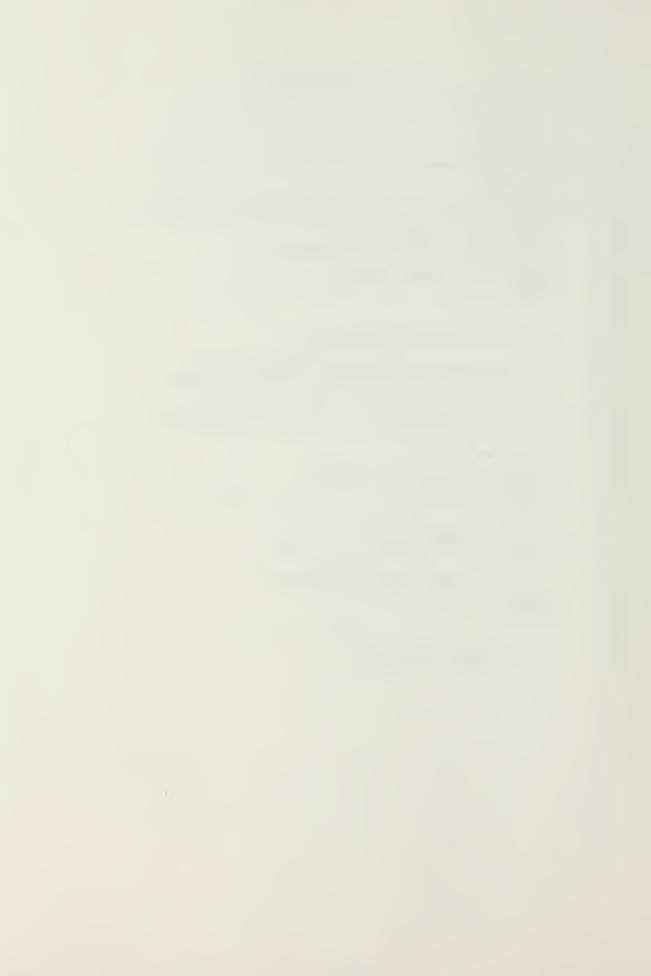
```
simulator
              Page 8
                        Fri Feb 8 07:10:27 1980
421 C
                      WHEN THIS HAPPENS, REPLACE ANY REMOVED
422 C
                      TOKENS AND RETURN IERROR=1.
423 C
424
           COMMON/EVENT/IEVENT(400,6), NXTEVT
           COMMON/TRANS/ITRANS(400,7), NXTTRN
425
           COMMON/IOTAB/IOTABL (4000), NXTTRE
426
427
428
           IPT=ITRANS(NUMBER, 2)
429
           KOUNT=IOTABL (IPT)
430
           IERROR=0
431
           DO 1000 I=IPT+1, IPT+KOUNT
432
433
           NEVENT=IOTABL(I)
434
           J = I
435
           IEVENT (NEVENT, 2) = IEVENT (NEVENT, 2) -1
           IF(IEVENT(NEVENT, 2).LT.0) GO TO 2000
436
437
     1000 CONTINUE
438
           RETURN
439
440
    2000 CONTINUE
441
           DO 3000 I=IPT+1,J
442
          NEVENT=IOTABL(I)
443
     3000 IEVENT(NEVENT, 2) = IEVENT(NEVENT, 2) +1
/1 /1 /1
           IERROR=1
445
446
          RETURN
447
          END
448
449
450
          SUBROUTINE MARKEM (NUMBER)
451 C
452 C
               MARK ALL OUTPUT EVENTS OF TRANSITION # 'NUMBER'
453 C
          COMMON/EVENT/IEVENT(400,6), NXTEVT
454
          COMMON/TRANS/ITRANS(400,7), NXTTRN
455
456
          COMMON/IOTAB/JOTABL(4000), NXTTRE
457
458
          IPT=ITRANS(NUMBER, 3)
          KOUNT=IUTABL (IPT)
459
460
          DO 1000 I=IPT+1, IPT+KOUNT
461
          NEVENT=IOTABL(I)
462
     1000 IEVENT(NEVENT, 2) = IEVENT(NEVENT, 2) +1
463
          RETURN
464
          END
465
          SUBROUTINE SCANR
466
467
          BYTE INDRD, ISC, IBLANK
468
          COMMON/SCAN/IWORD(15,10), NUMBER
469
470
          BYTE NHUFFR
471
          COMMON/SCAN1/NSUFFP(80)
472
          DATA ISC/1H;/
          DATA IBLANK/1H /
473
    0001 FORMAT(80A1)
474
475
          READ(1,0001,END=9999,ERR=9999) (NBUFFR(I),I=1,80)
476
          IPOINT=1
                SET POINTER TO FIRST CHAPACTER IN THE BUFFER
477 C
                NON PROCESS THE FIRST 15 TOKENS DELIMITED BY EITHER
478 C
479 C
                A BLANK (OR MULTIPLE BLANKS) OR A SEMICOLON.
480 C
```



```
Page 9 Fri Feb 8 07:10:27 1980
simulator
          DO 0002 NUMBER=1,15
482
          IFLAG=0
483 C
          SET INORD(NUMBER, X) = IBLANK (SET WORD TO ALL BLANKS)
          00 0003 I=1,10
484
485 0003 InORD(NUMBER, I) = IBLANK
486 C
          START SCANNING LINE FROM POINTER ON TO FIND NON-BLANK
487
          KOUNT=1
488 C
          "KOUNT" KEEPS TRACK OF THE NO. OF CHAR. IN THE TOKEN
489
          DO 0004 KPOINT=IPOINT,80
490
          IF(NBUFFR(KPOINT).NE.IBLANK .AND. NBUFFR(KPOINT).NE.ISC)
491
                      GO TO 0005
492
          IF(IFLAG.EQ.0) GO TO 0004
493
          IF(IFLAG.EG.1) GO TO 0006
494
    0005 CONTINUE
495
          IFLAG=1
          IWORD (NUMBER, KOUNT) = NBUFFR (KPOINT)
496
497
          KOUNT=KOUNT+1
498
          IF (KOUNT.GT.10) GD TO 0006
499 0004 CONTINUE
500
501 0006 CONTINUE
502 C
         END OF TOKEN FOUND, RESET SOME POINTERS
503
          IPOINT=KPOINT+1
504
          IF(IPOINT.GT.80) GO TO 0010
505
    0002 CONTINUE
506
507 C
          END OF BASIC TOKEN GETTING LOOP
508
509 0010 NUMBER=NUMBER-1
510
          PETURN
    9999 CONTINUE
511
          END OF FILE OR I/O ERROR DETECTED
512 C
513
     9998 FORMAT(' EOF OR ERROR ON SCANNER INPUT FROM UNIT 1')
          TYPE 9998
514
          NUMBER=0
515
          RETURN
516
517
          END
518
519
520
          SUBROUTINE XINTGR(NWORD, IVALUE)
521
522
              CONVERT THE ENTRY IN "IWORD" TO INTEGER
523 C
524 C
                 RETURN INTEGER "IVALUE"
525
          BYTE IWORD
526
          COMMON/SCAN/IWORD (15,10), NUMBER
527
          BYTE TSTRNG
528
529
          DIMENSION TSTRNG(10)
530
          BYTE IBLANK
          DATA IBLANK/1H /
531
          DO 0001 I=1,10
532
533
          KOUNT=I
534
          TSTRNG(I)=IWORD(NWORD,I)
          IF (IWORD (NWORD, I) . ED . IBLANK) GO TO 1000
535
    0001 CONTINUE
536
537
    1000 CONTINUE
538
          KOUNT=KOUNT-1
539
540
    2004 FORMAT(X, 10A1)
```



```
Page 10
                        Fri Feb 8 07:10:27 1980
simulator
     2005 FORMAT(X,1110)
541
542
          DO 2000 I=1,KOUNT
543
          J = 11 - I
544
          K=(KOUNT+1)-I
545
     2006 TSIRNG(J)=TSTRNG(K)
546
          L=10-KOUNT
547
          DG 2007, I=1,L
548
     2007 TSTRNG(I)=IBLANK
          OPEN (UNIT=2, NAME='ISTORE', TYPE='NEW', INITIALSIZE=20)
549
550
          WPITE(2,2004) (TSTRNG(I), I=1,10)
551
          CLOSE (UNIT=2, DISPOSE='SAVE')
552
          OPEN(UNIT=2, NAME='ISTORE', TYPE='OLD')
553
          READ(2,2005) IVALUE
554
          CLOSE (UNIT=2, DISPOSE='DELETE')
555
          RETURN
556
          END
557
558
559
          FUNCTION MATCHS (NUMB, STRING, NCHAR)
560
               THIS FUNCTION DETERMINES IF SCANNER TOKEN
561 C
               INOPD(NUMB) MATCHES THE CHARACTERS IN "STRING"
562 C
              AT LEAST FOR THE FIRST "NCHAR" CHARACTERS.
563 C
564
               IF THERE IS A MATCH, IT RETURNES THE INTEGER "1"
565 C
566 C
              NO MATCH RETURNS "0".
567
568
          RYTE IWORD
569
          COMMON/SCAN/IWORD (15, 10), NUMBER
          BYTE STRING
570
571
          DIMENSION STRING(10)
572
          MATCHS=0
573
574
          DO 0001 I=1,NCHAR
          IF (IWOPD (NUMB, I).NE.STRING(I)) RETURN
575
576
    0001 CONTINUE
577
578 C
           IF YOU GET HERE, THEY WERE THE SAME ...
          MATCHS=1
579
580
          RETURN
581
          END
582
583
          END OF PROGRAM SIMULATOR
584 C
```



```
transgraph.c Page 1 Fri Feb 8 05:01:00 1980
 1 #
 5
        3
 4
 5
        /*******
                          PROGRAM TRANSGRAPH.C
 6
        /******
 7
        /******
 8
        /*****
                     STEPHEN C. JENNINGS JC91 USMC
 9
        /******
                       ROBERT J. HARTEL CS91 USA
10
11
                       WRITTEN FALL QUARTER 1979
12
        /******
                       NAVAL POSTGRADUATE SCHOOL
        /******
                         MONTEREY, CALIFORNIA
13
14
        /******
15
        /*********************
16
17
18
19 /******************
20 /** EXTERNAL DECLARATIONS **/
21 /****************
55
23
24 /*** LITERALS ***/
25
26 #define: header
27 #define
                     50
          pictures
28 #define
                     100
           bounds
29 #define limit
                    500
30
31
32 /*** STRUCTURES ***/
33
34 struct {
                           /* data structure information on net nodes .. */
35
36
      int ctrll;
                       /* store control char not used in program ... */
37
      int nameptr;
                       /* index to names array ..... */
                       /* initial marker state of the network ..... */
38
      int marker;
39
                       /* x cordinate of place ..... */
      int xcord;
40
      int ycord;
                       /* y cordinate of place ..... */
41
      int plot;
                       /* whether or not place is to be plotted .... */
42
                       /* length of name associated with place .... */
      int length;
43
44 }file1 [bounds], *hp1;
                          /* pointer into data structure ...... */
45
46
47 struct 1
                           /* data structure information on transitions. */
48
49
      int ctrl2;
                       /* store control char not used in program ... */
50
                       /* index to names array ..... */
      int trnptr;
      int intrn;
51
                       /* pointer to inputs for a transition ..... */
52
                       /* pointer to outputs for a transition ..... */
      int outtrn;
53
                       /* x cordinate of transition ..... */
      int xxcord;
54
                       /* y cordinate of transition ...... */
      int yycord;
                       /* whether or not transition is to be plotted */
55
      int trnplot;
56
      int trnlen;
                       /* length of name associated with transition */
57
58 }file2 [bounds], *bp2; /* pointer into data structure ....... */
59
60
```



```
Fri Feb 8 05:01:00 1980
transgraph.c
             Page 2
 61 /*** INTEGERS ***/
 62
 63 int al,a2,a3,a4;
                            /* global storage for each data structure ... */
64 int buffer[bounds];
                            /* buffer into which each frame is read .... */
65 int cntrl[1];
66 int ctroverflow;
                            /* variable containing # transitions fired .. */
                            /* keep track of nodes overflow status ..... */
 67 int dfltcolor;
                            /* a default color for indicating overflows . */
 68 int fdfbuf;
                            /* file descriptor for RUN... files ..... */
 69 int fdabuf;
                            /* file descriptor for RUN..Z files ..... */
 70 int firing[bounds];
                            /* storage into which fired places are read . */
                            /* counter passed to a function ...... */
 71 int ictr;
 72 int ievents;
                            /* number of non- & displayable nodes ..... */
 73 int iflag;
                            /* counter for the interrupt mechanism ..... */
 74 int iotbl[limit];
                            /* forms input-to-output relationship ..... */
 75 int kpictures;
                            /* counter for the iterations of the network. */
 76 int link+b1[100][4];
                            /* version 1 % 2 screen nodes locations ..... */
77 int abrolot;
                            /* counter for number of displayed nodes .... */
                            /* store count fields for data structures ... */
78 int nbytes[2];
79 int overflowtb1(100)(2); /* data structure to store overflow locations */
80 int set;
                            /* user selected conrac graphics screen .... */
81 int sflag(20);
                            /* saves flag for later use by trnlite() .... */
82 int tblctr;
                            /* a counter for version 3..reset conditions. */
83 int uniquetol[100][4];
                            /* reset table locations for version 3 ..... */
 84 int vers; 1
                             /* user selected option ..... */
85
86
87 /*** CHARACTERS ***/
88
89 char fbuf[20];
90 char gbuf[20];
                            /* buffer to store name of second file ..... */
                          /* buffer to store name of third file..... */
/* character array for node labels ...... */
91 char names[limit];
 92 char scrn;
                            /* option variable for display to the screen. */
93
94
95 /*******************
96 /** FUNCTION MAIN **/
97 /********************
98
99 main() {
100
101
                       /* declare 'rubout' globally ......*/
       extern rubout();
102
       init();
                         /* read input file ..... */
                         /* verify if user wants to see data structure */
103
       determine();
       display();
select();
                         /* display input to crt ..... */
104
105
                         /* select version of simulation & genisco set */
       prepare(2);
106
                        /* draw network nodes on confac ..... */
107
       drawnode();
       places();
                        /* verify correct nooes grawn ...... */
108
109
                         /* function displays network transitions .... */
       trnslink();
110
                         /* starting status of network packets ..... */
       imark();
       signal(2,rubout); /* sets 'BRK' as interrupt ....... */
111
                         /* successive iterations of network flow .... */
112
       markina();
                        /* closing out oraphics facilities ..... */
113
       qnfini();
114
115 }
116
117
118 /******************
119 /** PROGRAM FUNCTIONS **/
120 /****************
```



```
Page 3 Fri Feb 8 05:01:00 1980
transgraph.c
121
122
123 pause(peroid) (
124
        /* function necessary as sleep() not compatible with signal() */
125
        int i,j,k;
126
        printf("***>>interrupt.....");
127
128
         for('i=0;i<persig;i++) {
129
             for(j=0;j<400;j++) {
130
                 for(k=0;k<1000;k++) {
131
                 }
132
             }
133
        printf("***>>wait.....");
134
135 return;
136 }
137
138
139 rubout() (
140
        /* function enables the 'brk' key as the interrupt signal */
141
        char halt;
142
143
        space(2);
144
        printf
        ("***>>> received signal...frame number %d...<ret> to continue \n",
145
146
        (iflaa+2));
147
        ("***>>> for termination of program...type 'brk' from console \n");
148
        while ((halt=detchar())!='\n') {
149
150
               /* do-nothing loop */
151
152
        signal(2, rubout);
153 return;
154 }
155
156
157 space(returns) {
158
         int i;
159
160
         for (i=0; i<returns; i++) {
             printf("\n");
161
162
163 return;
164 }
165
160
167 init() {
168
        /* function opens unformatted file % initializes start condition */
169
        int a, bufctr, count, fd, i, j;
170
        char cbuf[20],c,f;
171
172
        space(2);
        printf("***->TRANSGRAPH ILLUSTRATES PETRINET SIMULATION MODELS");
173
174
        space(2);
175
        printf("***->ENTER THE NAME OF THE FILE TO BE PROCESSED..BUT \n");
        printf("
                     NOTE THAT THIS FILE MUST BE AN UNFORMATTED FILE NO");
176
177
        printf("
                      PRODUCED AS A RESULT OF EXECUTING simulator.out \n");
178
        printf("
                     THE LAST LETTER OF WHICH MUST END IN LETTER 'X' \n");
179
      error:space(2);
        printf("***->");
180
```



```
transgraph.c
                 Page 4
                         Fri Feb 8 05:01:00 1980
181
182
         i = 0:
183
         while((c=getchar()) != '\n') {
184
              cbuf[i]=c;
185
              i++;
186
187
        cbuf-[i] = '\0';
138
189
        bufctr=i;
190
         for(j=0;j<bufctr;j++) {
191
             qouf[j]=fbuf[j]=chuf[j];
192
             if(cbuf[j]=='X') {
193
                 fbuf[j]='Y';
                 gbuf[j]='Z';
194
195
                 qbuf[j+1]=fbuf[j+1]='\0';
196
                 j=bufctr;
197
             -}
198
        }
199
200
        fd = open(cbuf,0);
201
        if (fd <= 0) {
            printf("***->error occurred in opening file...try again");
505
203
             space(3);
204
             goto error;
205
206
207
        if((count = read(fo, nbytes, header)) != header)
805
            orintf("error occurred in nbytes read");
209
        ievents = (nbytes[1] - 1);
210
        al = nbytes[1];
        a = (nbytes[1]-1) *14;
211
212
        if((count = read(fd,file1,a)) != a)
213
             printf("error occured in file1 read");
214
215
        if((count=read(fd,nbytes,header))!=header)
216
            orintf("error occurred in nbytes read");
217
        a2=nbytes[1];
218
        a=(nbytes[1]-1)*16;
219
        if((count=read(fd,file2,a))!=a)
550
            printf("error occurred in file2 read");
221
555
        if((count=read(fd,nbytes,header))!=header)
            printf("error occurred in header read");
553
224
        a3=nbytes[1];
        a=(nbytes[1]+1)*2;
225
226
        if((count=read(fd,iotbl,a))!=a)
227
            printf("error occured in iothl read");
855
229
        if((count=read(fd,nbvtes,header))!=header)
230
            orintf("error occurred in header read");
231
        a4=nnytes[1];
232
        a=nbytes[]]+1;
233
        if((count=read(fd,names,a))!=a)
            printf("error occurred in names read");
234
235
        close(fd);
236
237 return;
238 }
239
240
```



```
Page 5
                           Fri Feb 8 05:01:00 1980
transgraph.c
241 determine() {
242
        int i:
243
        char d, dbuf (20);
244
245
        space(2);
246
        printf("***->FUNCTION 'DETERMINE' ALLOWS THE USER \n");
        printf(" TO EXAMINE ALL PRIMARY DATA STRUCTURES");
247
248
       over:space(2);
        printf("***->IF THIS FEATURE IS DESIRED TYPE 1 IF NOT 0,...<RET>");
249
250
        space(2);
251
        printf("***->");
252
253
        i=0;
        while((d=getchar())!='\n') {
254
255
            dbuf[i]=d;
256
             i++;
257
        3
258
        dbuf[i]='\0';
259
560
        i=0;
        while(dbuf[i]!='\0') {
261
262
            d=abuf[i];
263
             switch(d) {
264
                 case'0':
265
                      scrn='0';
266
                      break;
                case'1':
267
268
                      scrn='1';
593
                      printf("***->USE CONTROL Q WHEN SCREEN FULL");
270
                      break;
271
                 default:
272
                      printf("***->either blank or invalid entry");
273
                      goto over;
274
                      break;
275
            }
276
            1++;
277
        }
278 return;
279 }
280
281
282 display() {
283
     int i;
284
285
      if(scrn=='1') {
286
          space(2);
287
          bpl = filel;
288
          printf("***=> FILE1 DATA STRUCTURE");
280
          space(2);
290
                                                                   length \n");
          printf("Infeed
                                                            plot
                           =
                                 marker
                                           xcord
                                                    ycord
291
          for (i=0;i<al; i++)(
595
              printf("%d \t %d \t %n",
293
                  bol->ctrli,brl->namentr, bol->marker,bol->xcord,
294
                  hel->ycord,bel->plot,bel->length);
295
              hp1++;
296
          }
297
298
          space(2);
299
          bp2 = file2;
          printf("***-> FILE2 DATA STRUCTURE");
300
```



```
Page 6
                        Fri Feb 8 05:01:00 1980
transgraph.c
301
          space(2);
302
          printf
          ("Infeed
303
                    tropto intro auttro xxcord yycaru troplat trolen \n");
304
          for (i=0;i<a2; i++){
305
              printf("%d \t %d \t \n",
306
                  hp2->ctrl2,bp2->trnptr,bo2->intrn,bp2->outtrn,
                  bo2->xxcord,bo2->yycord,bo2->trnplot,bp2->trnlen);
307
308
              bp2++;
309
          }
310
311
          space(2);
          printf("***-> IOIBL DATA ARRAY");
312
313
          space(2);
314
          for (i=0; i<a3; i++) {
315
              printf("%d ", iotbl[i]);
316
317
318
          space(2);
          printf("***-> NAMES DATA ARRAY");
319
320
          scace(2);
321
          for (i=0; i<a4; i++) (
             printf("%c",names[i]);
322
323
324
          space(2);
325
     }
326 return;
327 }
328
329
330 prepare(type) {
        /* function designates set, screen size and color table */
331
332
        int notoy;
333
334
        y=0;
335
        genisco (set);
336
        erase();
        screen(0.0,0.0,511.0,511.0);
337
338
        setmod(type);
339
        coltab();
340
        calart(11);
        for(n=12;n<14;n++) {
341
342
            color(n);
343
            for(t=0;t<512;t++) {
344
                segmnt(0,y+t,511,y+t);
345
346
        }
347 return;
348 }
349
350
351 drawnode() (
        /* function displays type % location of network nodes */
352
        char c.*nptr,hold;
353
354
        int a,b,c1r1b1,count,d,entry,h,i,j,k,l,test,x,y,*z;
355
        float s;
        double sqrt();
356
357
358
       bol = filel;
359
        a=0;
        count = 1;
360
```



```
Page 7
                           Fri Feb 8 05:01:00 1980
transgraph.c
361
362
        while((test=(bp1->nameptr))!=0) {
363
            color(14);
364
             test++;
365
             z = &names[test];
366
             c = *z;
367
368
             if ((b=(pp1->plot))!= 0) {
369
                 switch (c){
370
                     case 'I':
                          x = (bp1->xcord);
371
                          y = (r_1->ycord);
372
373
                          for(d=0;d<10;d++) {
374
                              seamnt(x=16,y+2+d,x,y+2+d);
375
376
                          linktbl(a) (3) = 1;
377
                          breaki
                     case '0':
378
                         x = (bp1->xcord);
379
380
                          y = (hp1->ycord);
381
                          for(d=0;d<10;d++) {
                              segmnt(x-16,y-2-d,x,y-2-d);
382
383
                          3
384
                          linktbl[a] [3] =1;
385
                          break;
                     case 'R':
386
                          x = (bn! - > xcord);
387
                          y = (hp1->ycord);
388
389
                          for(d=0;d<31;d++) {
390
                           segmnt
                           (x-18+d/4,y+15-d,x+18-d/4,y+15-d);
391
392
393
                          ciribl=14;
                          label(x,y,test,clrlbl);
394
395
                          break;
                     case 'S':
396
                          x = (bp1->xcord);
397
                          y = (hol->ycord);
398
                          for(d=0;d<31;d++) {
399
                            seamnt(x=18,y=15+d,x+18,y=15+d);
400
401
                          if(vers==3) {
402
403
                              notr= &names[test+1];
404
                              hold= *nptr;
                              clr1bl=atoi(3hold);
405
                              label(x, v, test, clrlbl);
406
                              color(14);
407
                          }
408
409
                          else (
                               c1r1b1=14;
410
                               label(x,y,test,clrltl);
411
                          }
412
                          break;
413
                      case 'I':
414
                          x = (bol -> xcord);
415
                          v = (bol = > v cord);
416
                          for (k=0;k<19;k++) {
417
418
                              s=k;
                              i=(x-(sqrt(324.-s*s)));
419
                              j=v+g;
420
```



```
Fri Feb 8 05:01:00 1980
transgraph.c
                Page 8
421
                               1=(x+(sqrt(324.+s*s)));
422
                               seamnt(i,j,1,j);
423
                           }
424
                           for (k=0;k<19;k++) {
425
                               s=k;
426
                               i = (x - (sart(324. - s*s)));
427
                               j=y-s;
428
                               1=(x+(sart(324.-s*s)));
429
                               seamnt(i,j,l,j);
430
431
                           if(vers==3) {
432
                               nptr = %names[test+1];
433
                               hold = *nptr;
434
                               clrlbl=atoi(&hold);
435
                               label(x,y,test,clrlbl);
436
                               color(14);
437
438
                           else {
439
                                c1r1b1=14;
440
                                label(x,v,test,clrlbl);
441
                           }
442
                           break;
443
                      default:
444
                           printf("name not valid identifier");
445
                           space(2);
446
                           break;
447
448
449
             linktbl[a][0]=count;
450
             linktbl[a][1] = x;
451
             linktbl[al[2] = y;
452
             a++;
453
454
         count++;
455
         bp1++;
456
457
         if(vers==3) {
458
             entry=0;
459
             uniquetb1[entry][0]=linktb1[0][0];
             uniquetbl [entry] [1] = linktbl [0] [1];
460
461
             uniquetbl [entry] [2] = linktpl [0] [2];
462
             uniqueth1[entry][3]=linktb1[0][3];
463
             tblctr=1;
464
             for(i=0;i<a;i++) {
465
                     if(uniquern) [entry] [1] == linkth] [i+1] [1] 88
466
                         uniqueth1 [entry] {2} == linkth1 (i+1) (2)) {
                          /* do nothing */
467
408
                     }
459
                     else (
470
                         entry++;
471
                         uniquetol[entry][0]=linktbl[i+1][0];
                         uniqueth1 (entry) []] = linktb] [i+1] [];
472
                         uniquethl [entry] [2] = linktbl [i+1] [2];
473
                         uniquetbl [entry] [3] = linkthl[i+1] [3];
474
                         tblctr++;
475
476
477
478
479
        nbrolot=a;
480 return;
```



```
transgraph.c
                 Page 9
                           Fri Feb 8 05:01:00 1980
481 }
432
483
484 places() {
485
        int h,i,j,k;
486
487
         if(strn=='1') {
488
             space(2);
             orintf("***->DATA STRUCTURE LINKTBL ");
489
490
             snace(2);
491
             for(i=0;i<nbrolot;i++) {
                 for(h=0;h<4;h++) {
492
493
                     printf("%d --",linktbl(i)(h));
494
495
                 space(1);
496
497
             space(2);
498
             if(vers==3) {
                 printf("***->DATA STRUCTURE UNIQUETBL");
499
500
                 space(2);
                 for(j=0;j<tblctr;j++) {</pre>
501
502
                     for(k=0;k<4;k++) {
503
                          printf("%d --",uniquetbl(j)[k]);
504
                     }
505
                     space(1);
506
             }
507
508
             space(2);
509
        }
510 return;
511 }
512
513
514 label(xx,yy,zz,clbl) {
515
           /* determines node lanel placement in relation to node */
516
            int f,i;
517
            char a;
518
519
            color(clb1);
520
            if(xx>250) {
521
                if(yy>250) a='1';
522
                else a='2';
523
524
            else (
525
                if(yy>250) a=131;
526
                olse a='4';
527
528
            f=(bol->lenath);
529
530
            switch (a) {
                case'1':
531
                    for(i=0;i<2;i++) {
532
                       charac((xx+(20+(8*i))),yy+15,names(zz));
533
534
                       ZZ++;
535
                    )
536
                    break;
                case'2':
537
                    for(i=0;i<2;i++) {
538
                       charac((xx+(20+8*i)),yy=28,names(zz]);
539
540
                       22++;
```



```
transgraph.c
               Page 10
                         Fri Feb 8 05:01:00 1980
541
                    }
542
                    break;
                case'3':
543
544
                    for(i=0;i<2;i++) {
545
                       charac((xx-(32-(8*i))),yy+15,names(zz));
546
547
                    }
548
                   break;
               case'4':
549
550
                    for (i=0;i<2;i++) {
551
                        charac((xx-(32-8*i)),yy-26,names[zz]);
552
                        22++;
553
554
                    break;
555
           }
556 return;
557 }
558
559
560 select() {
561
      int i,n;
        char v, vbuf [20];
562
563
564
        space(2);
        orintf("***->THERE ARE 3 VERSIONS TO THIS GRAPHICS PACKAGE \n");
565
        printf("
566
                    PLEASE SELECT ONE OF THE FOLLOWING VERSIONS: \n");
567
      again:space(2);
       printf("
                       VERSION 1 ... PETRI-NET PACKAGE ..... TYPE 1 \n");
568
        printf("
569
                       VERSION 2 ... PACKET PEPRESENTATION ... TYPE 2 \n");
        printf("
570
                       VERSION 3 ... MULTIROUTING PACKAGE .... TYPE 3 \n");
571
        n = 0;
572
      twice:snace(2);
        printf("***->");
573
574
        n++;
575
        i = 0;
576
        while ((v=getchar()) != '\n') {
577
            vcuf[i] = v;
578
            1++;
579
        vbuf[i] = '\0';
580
581
582
        i = 0;
        while(vouf[i] != '\0') {
583
584
            v = vouf[i];
585
            if(n==1) {
586
                  switch(v) {
587
                     case '1':
588
                      vers = 1;
589
                     break;
                     case '2':
500
591
                      vers = 2;
592
                      breaki
593
                     case '3':
594
                      vers = 3;
595
                     break;
596
                     default:
597
                     printf
598
                     ("***->incorrect version try again!");
599
                     goto adain;
600
                     break;
```



```
Page 11
transgraph.c
                           Fri Feb 8 05:01:00 1980
601
602
                   i++;
603
             }
604
             else {
605
                   switch(v) {
                     case '0':
606
607
                      set = 0;
608
                      break;
609
                     case '1':
610
                      set = 1;
611
                      break;
612
                     case '2':
613
                      set = 2;
614
                      break;
615
                     default:
616
                      printf
                      ("***->incorrect genisco set try again!");
617
618
619
                      ("
                             set selection should be 0,1 or 2");
620
                      n = 1;
621
                      noto twice;
655
                      break;
623
                   }
624
                   j++;
            } !
625
626
627
         if(n==1) {
628
              space(2);
629
              orintf("***=>NOW SELECT THE GENISCO SET YOU WISH \n");
                         THE PROGRAM TO BE DISPLAYED TO .... \n");
630
              printf("
              crintf("
631
                           IN C3 LAB EITHER SETO, SET1 OR SET2 \n");
              goto twice;
632
        }
633
634 return;
635 }
636
637
638 imark() {
639
        /* marks initial state of system by calling appropriate function */
640
        int b.colour.e.a.x.y;
641
642
        bol = filel;
643
        dfltcolor=3;
644
        color(dfltcolor);
645
        colour=2;
646
        ctroverflow=0;
647
        while ((a=(bn1->nameptr)) !=0) {
648
            if((b=(bn1->plot)) !=0) {
                    switch(vers) (
649
650
                        case 1:
651
                            ivers1();
652
                            break;
653
                        case 2:
654
                           ivers2(colour);
655
                           break;
656
                        case 3:
                            ivers3(colour);
657
658
                           break;
659
                    }
660
```



```
Page 12
transgraph.c
                           Fri Feb 8 05:01:00 1980
601
            if(dfltcolor!=3) color(3);
662
            bo1++;
663
        -}
664
        color(14);
665
        orinta(0,350.,480.,"TIME FRAME = 1");
666
        preread(1);
667
        trnlite(1);
668
        displa();
        hold();
669
670
        trnlite(2);
671
        reset(1);
672
        color(13);
673
        ovrflow();
674
        orintg(0,350.,480.,"TIME FRAME = 1");
675 return;
676 }
677
678
679 hold() {
680
        int holding;
681
682
        space(2);
683
        printf("***->THIS IS THE INITIAL STATE OF THE NETWORK \n");
        printf(" TYPE <RETURN> TO CONTINUE EXECUTION.... \n");
684
685
        while((holding=getchar())!='\n') {
686
            /* do nothing loop */
687
        }
688 return;
689 }
690
691
692 ivers1() {
693
       int e,x,y,z;
694
        char check;
695
696
        e=(bp1->marker);
697
        x=(bp1->xcord);
698
        v=(bol->ycord);
699
        z=(bpl->namentr);
700
        if((check=names(z+1))!='I' && (check=names(z+1))!='0') {
701
             printg(0,x-3.0,511.-(y-3),"%d",e);
702
        }
703
        else {
704
             if(check=names{z+1}=='I') printa(0,x-14.0,511.-(y+2),"%d",e);
705
             else printq(0,x-14.0,511.-(y-9),"%d",e);
706
        }
707 return;
708 }
709
710
711 ivers2(colour) {
712
       int e.x.v.zi
713
        char check;
714
715
        e=(hol->marker);
        x=(bo1->xcord);
716
717
        v=(bol->ycord);
        z=(bo1->namentr);
718
        if((check=names[z+1])!='I' && (check=names[z+1])!='0') {
719
720
            pckt2(x, y, e, colour);
```



```
transgraph.c
                Page 13
                           Fri Feb 8 05:01:00 1980
721
722
        else (
              if(check=names(z+1)=='I') printg(0,x=14.0,511.-(y+2),"%a",e);
723
              else printg(0,x-14.0,511.-(y-9),"%d",e);
724
725
        }
726 return;
727 }
728
729
730 ivers3(colour) {
731
        char keep, *kptr;
732
        int a,aa,b,bb,c,cc,clr[251,i,k,n,stack,total,x,y;
733
734
        total=0;
735
        n=1;
736
        x=(bp1->xcord);
737
        y=(bpl->ycord);
738
        a=(bpl->length);
739
        b=(bol->nameptr);
740
        c=(bol->marker);
741
        total=total+c;
742
743
        if(c>0) {
744
             far(i=0;i<c;i++) {
745
                 kotr = %names[b+a-2];
746
                keep = *kptr;
747
                 clr[n] =atoi(&keep);
748
                 n++;
749
750
        }
751
752
        kotr = &names(b+a);
753
        keep = *kotr;
754
        stack=atoi(&keep);
755
        if(names[b+(a=1)]!='0') {
756
             if(names(o+(a-1)) == '2') stack=stack+20;
757
             else stack = stack + 10;
758
759
760
        for(i=0;i<stack=1;i++) {
761
            bp1++;
762
763
             aa=(bo1->length);
764
             bn=(bol->nameotr);
765
             cc=(hol->marker);
             total=total+cc;
766
767
             if(c,c>0) {
768
                 for(k=0;k<cc;k++) {
769
                     kntr = &names(bb+aa=2);
                     keep = *kptr;
770
771
                     clr[n]=atoi(kkeep);
772
                     n++;
773
                 )
774
775
        if(names(b+1)!='I' %% names(b+1)!='0') {
776
777
          ockt3
          (x, y, total, clr[1], clr[2], clr[3], clr[4], clr[5], clr[6], clr[7],
778
779
           colour);
780
```



```
Page 14 Fri Feb 8 05:01:00 1980
transgraph.c
781
        else {
            if(names[b+1] == 'I') printg(0,x-14.0,511.-(y+2),"%d",total);
782
783
            else printq(0,x-14.0,511.-(y-9),"%d",total);
784
785 return;
786 }
787
788
789 preread(flag) {
790
         int bucket[2], count, fd, fq, i, nbrtrns;
791
         if(flag!=3) {
        if(flag==1) {
792
793
            fd = open (fbuf, 0);
794
            if (fd<=0) {
795
                printf("***->error occurred in opening fd file");
796
797
            fdfbuf=fd;
798
            fa = open (abuf, 0);
799
            if (fa<=0) {
800
                printf("***->error occurred in opening fg file");
801
802
            fdgbuf=fg;
803
804
        if((count=read (fdfbuf, bucket, 2))!=2) {
805
            printf("***->error occurred in fd bucket read");
806
        if((count=read (fdfbuf, buffer,(ievents*2)))!=(ievents*2)) {
807
808
            printf("***->error occurred in buffer read");
809
810
        if((count=read (fddbuf, bucket, 2))!=2) {
            printf("***=>error occurred in fg bucket read");
811
812
813
        if((count=read (fdgbuf, cntr1,2))!=2) {
814
            printf("***->error occurred in cntrl read");
815
816
        if(cntr1(0) == 0) {
817
            space(2);
813
            printf("***->the last network state has been achieved");
819
            kpictures=pictures+1;
820
            space(2);
821
        }
855
        else {
            if((count=read (fdqbuf, bucket, 2))!=2) {
823
                  printf("***=>error occurred in bucket read");
824
825
826
            nortrns = chtrl[0] *2;
827
            if((count=read (fdanuf, firing,nhrtnns))!= nbrtnns) {
                  orintf("***->error occurred in firing read");
828
829
            }
830
        }
831
        }
832
        else {
833
        close(fdfbuf);
834
        close(fdaouf);
835
        }
836 return;
837 }
838
839
840 stage() {
```



```
Page 15
transgraph.c
                           Fri Feb 8 05:01:00 1980
841
        if (kpictures!=0 && kpictures<pictures+1) {
842
843
            preread(2);
844
             if(kpictures!=pictures+1) {
845
                trnlite(1);
                displa();
846
847
                pause(1);
848
                 trnlite(2);
849
            }
850
        }
851 return;
852 }
853
854
855 marking() {
856
        /* function displays successive iterations of the network */
857
        int colour, draw, i, mark, n, x, y;
858
859
        bol = filel;
860
        n = 2;
861
        /* following loop processes levent # oata entries each pass */
802
863
        for(kpictures=0;kpictures<pictures;kpictures++) {
864
            stage();
865
            if(kpictures>0) {
866
                reset();
867
                color(13);
868
                ovrflow();
869
                printg(0,350.,480., "TIME FRAME = %d",n);
870
                n++;
871
            iflag = kpictures;
872
873
            (tofa<-fad) = went
874
            dfltcolor=3;
875
            (Tofostift) Tofos
876
            colour=2;
877
            ctroverflow=0;
878
             for (i=0; i < ievents; i++) {
879
                 if (draw == 1) {
                     ictr = i;
890
                     switch (vers) (
881
882
                         case 1:
                             versi();
883
884
                             break;
885
                         case 2:
886
                             vers2(colour);
887
                             nreak;
                         case 3:
888
889
                             vers3(colour);
890
                             breaki
                     }
891
892
                }
893
                 i = ictr++;
894
                 if(dfltcolor!=3) cclor(3);
895
                501++;
                draw = (bpl -> plot);
896
897
898
            dfltcolor=14;
            color(dfltcolor);
899
900
            colour=13;
```



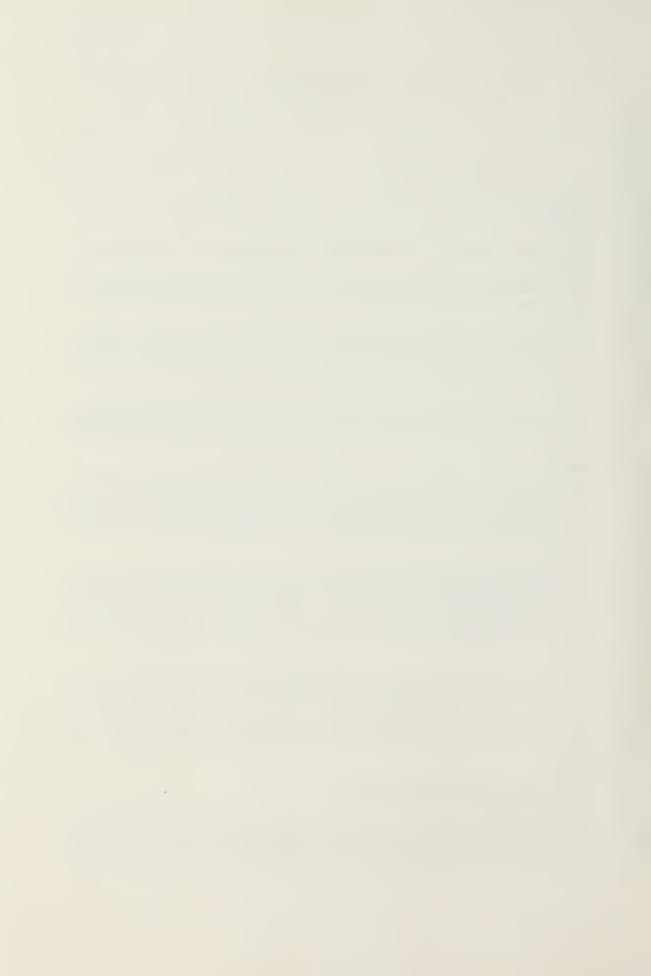
```
transgraph.c
                 Page 16
                             Fri Feb 8 05:01:00 1980
901
             printg(0,350.,480.,"TIME FRAME = %d",n);
902
             pause(2);
903
             trnlite(3);
904
             bp1 = file1;
905
         }
906
        preread(3);
907 return;
908 }
909
910
911 ovrflow() {
912
913
        int i,x,y;
914
915
        i=0;
916
        while(overflowtb1[i][0]!=0) {
917
             x=overflowtb1[i][0];
918
             v=overflowttl[i][1];
919
            block((x-3)+1.,511.-(y+28)+1.,(x+10)+1.,511.-(y+20)+1.);
920
            overflowtb1[i][0]=0;
921
             overflowtb1[i][1]=0;
922
             1++;
923
        }
924 return;
925 }
926
927
928 vers1() {
929
        int e.x.y.z;
930
        char check;
931
932
        color(3);
933
        e=bufferlictrl;
934
        x=(bol->xcord);
935
        y=(pp1->ycord);
936
        z=(bol->nameptr);
937
        if((check=names(z+1))!='I' && (check=names(z+1))!='0') {
938
             printg(0,x-3.0,511.-(y-3),"%d",e);
939
        }
940
        else {
941
              if(check=names(z+1)=='I') printq(0,x-14.0,511.-(y+2),"%d",e);
942
             else printg(0,x-14.0,511.-(y-9),"%d",e);
943
        -}
944 return;
945 }
946
947
948 vers2(colour) {
949
       int mark, x, y, z;
950
        char check;
951
952
        x=(bo1->xcord);
953
        y=(bo1->ycord);
954
        z=(nn1->namentr);
955
        mark=buffer(ictr);
        if((check=names(z+11)!='I' %& (check=names(z+1))!='0') {
956
957
        ockt2(x,y,mark,colour);
958
        }
959
        else {
         if((check=names(z+11)=='I') printq(0,x=14.0,511.-(v+2),"%d",mark);
960
```



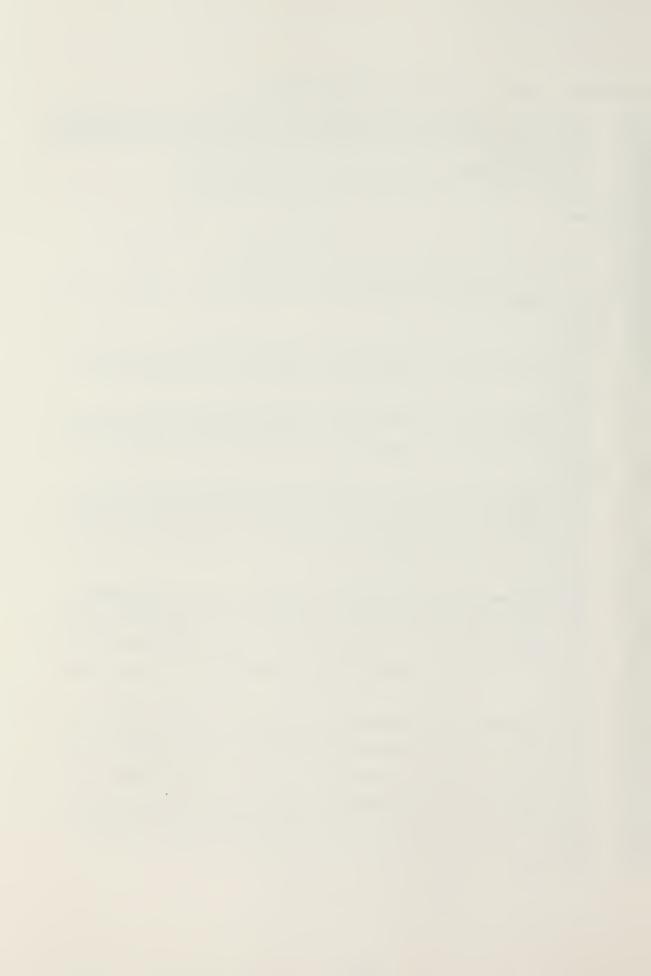
```
transgraph.c Page 17 Fri Feb 8 05:01:00 1980
961
         else printg(0,x-14.0,511.-(y-9),"%d",mark);
962
963 return;
964 }
965
966
967 vers3(côlour) {
968
        int a.aa,b,bb,mark,marks,clr(25),j,k,n,stack,total,x,y;
969
        char keep, *kptr;
970
971
        total=0;
972
        n=1;
973
        x=(bol->xcord);
974
        y=(bol->ycord);
975
        a=(bp1->length);
976
        b=(bp1->nameptr);
977
        mark=buffer(ictr);
978
        total=total+mark;
979
        if(mark>0) {
980
             for(j=0;j<mark;j++) (
981
                 kptr = %names(b+a-2);
982
                keep = *kptr;
983
                clr[n] = atoi(&keep);
984
                n++;
985
            1
986
        }
987
988
        kotr = %names[b+a];
989
        keep = *kotr;
990
        stack=atoi(&keep);
991
992
        if(names[p+(a-1)]:='0') {
            if(names(b+(a-1))=='2') stack=stack+20;
993
994
            else stack = stack + 10;
995
996
997
        for(j=0;j<stack-1;j++) {
998
            bp1++;
999
            ictr++;
1000
             aa=(bpl->length);
1001
             bb=(bol->nameptr);
1002
             marks=buffer[ictr];
1003
             total=total+marks;
1004
             if(marks>0) {
1005
                  for(k=0;k<marks;k++) {
1006
                     kptr = %names[bb+aa=2];
1007
                     keep = *kptr;
                     cirini=atoi(%keer);
100A
1009
                     n++;
1010
                 }
1011
             }
1012
1013
         if(names(b+1)!='I' 3% names(b+1)!='G') {
1014
1015
           (x,y,total,clr[1],clr[2],clr[3],clr[4],clr[5],clr[6],clr[7],
1016
            colour);
1017
         }
1018
         else (
             if(names[h+1] == 'I') printq(0,x-14.0,511.-(y+2), "%d",total);
1019
             else printg(0,x-14.0,511.-(y-9),"%d",total);
1020
```



```
transgraph.c
                Page 18
                            Fri Feb 8 05:01:00 1980
1021
         - 3
1022 return;
1023 }
1024
1025
1026 pckt2(xaxis,yaxis,point,class) {
1027
1028
      switch(point) {
        case 0:
1029
1030
         break;
1031
        case 1:
1032
         block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1033
         break;
1034
        case 2:
1035
         block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1036
         hlock((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1037
         break;
1038
        case 3:
1039
         block((xaxis=3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis=3)*1.);
1040
         block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1041
         block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1042
         break;
1043
        case 4:
1044
         block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1045
         block((xaxis=3)*1.,511.-(yaxis=7)*1.,(xaxis+3)*1.,511.-(yaxis=13)*1.);
1046
         block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1047
         clock((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
1048
         break;
1049
        case 5:
1050
         plock((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1051
         olock((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
         block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1052
1053
         block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
1054
         plock((xaxis=13)*1.,511.=(yaxis+3)*1.,(xaxis=7)*1.,511.=(yaxis=3)*1.);
1055
         break:
1056
        case 6:
1057
         block((xaxis=3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis=3)*1.);
1058
         block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1059
         block((xaxis=3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1060
         plock((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
1061
         block((xaxis-13)*1..511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
1062
         block((xaxis-13)*1.,511.-(yaxis+13)*1.,(xaxis-7)*1.,511.-(yaxis+7)*1.);
1063
         breaki
1064
        case 7:
1065
         olock((xaxis=3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis=3)*1.);
         nlcc<((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(vaxis-13)*1.);
1050
1067
         block((xaxis=3)*1.,511.=(yaxis+13)*1.,(xaxis+3)*1.,511.=(yaxis+7)*1.);
         plock((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
1058
1069
         plock((xaxis-15)*1.,511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
1070
         block((xaxis=13)*1.,511.=(yaxis+13)*1.,(xaxis=7)*1.,511.=(yaxis+7)*1.);
1071
         ologk((xaxis+7)*1.,511.-(yaxis+13)*1.,(xaxis+13)*1.,511.-(yaxis+7)*1.);
1072
         break;
1073
        default:
1074
         overflowth1 [ctroverflow] [0] =xaxis;
1075
         overflowtol(ctroverflow)[1]=yaxis;
1076
         ctroverflow++;
         plock((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1077
         block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1078
        block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(vaxis+7)*1.);
1079
         block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
1080
```



```
transgraph.c
                 Page 19
                            Fri Feb 8 05:01:00 1980
         block((xaxis=13)*1.,511.-(yaxis+3)*1.,(xaxis=7)*1.,511.-(yaxis=3)*1.);
1081
1082
         block((xaxis=13)*1.,511.-(yaxis+13)+1.,(xaxis=7)*1.,511.-(yaxis+7)*1.);
1083
         block((xaxis+7)*1.,511.-(yaxis+15)*1.,(xaxis+13)*1.,511.-(yaxis+7)*1.);
1084
         color(class);
1085
         dfltcolor=class;
1086
         printq(0,xaxis=3.0,511.-(yaxis+22)+3.0, "%d",point=7);
1087
1088
      - }
1089 return;
1090 }
1091
1092
1093
1094 ockt3(xaxis,yaxis,total,cl,c2,c3,c4,c5,c6,c7,class) {
1095
1096
      switch (total) {
1097
       case 0:
1098
         break;
1099
       case 1:
1100
         color(cl);
1101
         block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1102
         break;
       case 2:
1103
1104
         color(c1);
1105
         block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1106
         color(c2);
1107
         0 \log ((x_3x_1s_3)*1.,511.-(y_3x_1s_7)*1.,(x_3x_1s_3)*1.,511.-(y_3x_1s_13)*1.);
1108
         break;
1109
       case 3:
         color(cl);
1110
         block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1111
1112
         color(c2);
1113
         plock((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1114
         color(c3);
1115
         block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1116
         preak;
       case 4:
1117
1118
         color(cl);
1119
         block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1120
         color(c2);
         block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1121
1122
         color(c3);
1123
         nlock((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1124
         color(c4);
1125
         tlock((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
1126
         break;
1127
       case 5: ·
1128
         color(cl);
1129
         mlock((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1130
         color(c2);
         block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1131
1132
         color(c3);
         mlock((xaxis=3)*1.,511.=(yaxis+13)*1.,(xaxis+3)*1.,511.=(yaxis+7)*1.);
1133
1134
         color(c4);
         block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
1135
1136
         color(c5);
         block((xaxis-13)*1.,511.-(vaxis+3)*1.,(xaxis-7)*1.,511.-(vaxis-3)*1.);
1137
1138
         breaki
       case 6:
1139
1140
         color(cl);
```



```
Page 20
                            Fri Feb 8 05:01:00 1980
transgraph.c
1141
         block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1142
         color(c2);
1143
         block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1144
         color(c3);
1145
         block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1146
         color(c4);
1147
         block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
1148
         color(c5);
1149
         block((xaxis-13)*1.,511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
1150
         color(c6);
1151
         block((xaxis=13)*1.,511.-(yaxis+13)*1.,(xaxis=7)*1.,511.-(yaxis+7)*1.);
1152
         break;
       case 7:
1153
1154
         color(cl);
1155
         block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
         color(c2);
1156
1157
         block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1158
         color(c3);
1159
         block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1160
         color(c4);
1161
         block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
1162
         color(c5);
1163
         block((xaxis=13)*1.,511.=(yaxis+3)*1.,(xaxis=7)*1.,511.=(yaxis=3)*1.);
1164
         color(c6);
1165
         plock((xaxis=13)*1.,511.-(yaxis+13)*1.,(xaxis-7)*1.,511.-(yaxis+7)*1.);
1166
         color(c7);
1167
         block((xaxis+7)*1.,511.-(yaxis+13)*1.,(xaxis+13)*1.,511.-(yaxis+7)*1.);
1168
         break;
1169
       default:
1170
         overflowtb1[ctroverflow][0]=xaxis;
1171
         overflowth1 (ctroverflow) (1) =yaxis;
         ctroverflow++;
1172
1173
         color(c1);
1174
         block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
         color(c2);
1175
         block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1176
1177
         color(c3);
         block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1178
1179
         color(c4);
1180
         block((xaxis+7)*1.,511.-(vaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
1181
         color(c5);
         block((xaxis-13)*1.,511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
1182
1183
         color(c6);
         block((xaxis-13)*1.,511.-(yaxis+13)*1.,(xaxis-7)*1.,511.-(yaxis+7)*1.);
1184
1185
         color(c7);
         block((xaxis+7)*1.,511.-(yaxis+13)*1.,(xaxis+13)*1.,511.-(yaxis+7)*1.);
1185
1187
         color(class);
1188
         dfltcolor=class;
         printg(0, xaxis=3.0,511.-(yaxis+22)+3.0, "%d", tota1-7);
1189
1190
         break;
1191
       }
1192 return;
1193 }
1194
1195
1196 reset() {
        /* reset function for successive network iterations */
1197
1198
         int i, mark, x, y, z;
1199
1200
         if(vers==1 ;; vers==2) {
```



```
transgraph.c
                 Page 21
                           Fri Feb 8 05:01:00 1980
1201
                for(i=0;i<nbrplot;i++) {</pre>
1202
                 color(14);
1203
                 x=linktb1(i1(1);
1204
                 y=linktbl[il[2];
1205
                 z=1inktb1(i)(3);
1206 /
                 if(z==0) {
1207
                 block((x+3)*1.,511.-(y-7)*1.,(x+3)*1.,511.-(y-13)*1.);
1208
                 block((x-13)*1.,511.-(y+13)*1.,(x+13)*1.,511.-(y-3)*1.);
1209
1210
                 else {
1211
                 block((x-16)*1.,511.-(y-2)*1.,x*1.,511.-(y-10)*1.);
1212
                 block((x=16)*1.,511.=(y+10)*1.,x*1.,511.=(y+2)*1.);
1213
1214
1215
1216
         else {
1217
                for(i=0;i<tblctr-1;i++) {
1218
                 color(14);
1219
                 x=uniquetp1(il(1);
                 y=uniquetb1[i][2];
1220
1221
                 z=uniquetb1(i)(3);
1222
                 if(z==0) {
1223
                 block((x-3)*1.,511.-(y-7)*1.,(x+3)*1.,511.-(y-13)*1.);
1224
                 block((x-13)*1.,511.-(y+13)*1.,(x+13)*1.,511.-(y-3)*1.);
1225
1226
                 else {
1227
                     block((x-16)*1.,511.+(y-2)*1.,x*1.,511.-(y-10)*1.);
                     block((x-16)*1.,511.-(y+10)*1.,x*1.,511.-(y+2)*1.);
1228
1229
                 }
1230
                3
1231
         }
1232 returni
1233 }
1234
1235
1236 trnlite(tflag) {
1237
          int flag,h,i,ino,j,k,l,m,n,on,outo,otr,r,s,x,xx,y,yy;
1238
1239
          for(h=0;h<cntrl(0);h++) {
1240
         tp2=file2;
1241
         for(j=0;j<(firing(h)-1);j++) {
1242
             hn2++;
1243
1244
         if(on=(b02->trnplot)==1) {
1245
             x=(bp2->xxcord);
1240
              y=(bo2->vvcord);
              if(tflag==1 !; tflag==2) color(11);
1247
1248
             else color(15);
1249
              seamnt(x,y=20,x,y+20);
1250
              1=(ho2->trnlen);
1251
              k=(bp2->trnctr);
              for(i=0;i<1;i++) {
1252
                  charac((x-8:(8+i)),v+24,names(k:1]);
1253
1254
                  k++;
1255
             }
1256
               if(tflag==1 ;; tflag==3) {
1257
              if(tflag==1) color(11);
1255
1259
             else color(14);
1260
              flag=0;
```



```
Page 22
transgraph.c
                             Fri Feb 8 05:01:00 1980
1261
              m=(bo2->intrn);
1262
              r=iotbl(m);
              for(i=0;i<r;i++) {
1263
1264
                   ino=iotbl[m+1];
1265
                   for(j=0;j<norplot;j++) {</pre>
1266
                       if(inp==linktbl(j)(0)) {
1267
                           xx=linktbl[j][1];
1268
                           yy=linktbl[j][2];
1269
                            if(flag==0) {
1270
                                    if(xx<=x) flag=1;
1271
                                    else flag=2;
1272
                                    sflag[h]=flag;
1273
1274
                           if(flag==1) intrns1(xx,yy,x,y);
1275
                           else intrns2(xx,yy,x,y);
1276
                            j=nbrplot;
1277
                       }
1278
                  }
                  m++;
1279
1280
              }
1281
               - }
1282
1283
               it(tflag==2 | tflag==3) {
              if(tflag==2) color(11);
1284
1285
              else color(14);
1286
              n=(bp2->outtrn);
1287
              s=iotbl[n];
1288
              for(i=0;i<s;i++) {
1289
                  outp=ictbl(n+1);
1290
                   for(j=0;j<nhrplot;j++) {</pre>
1291
                       if(outn==linktb)[j]{0}) {
1292
                           xx=linkthl[j][11;
1293
                           yv=linktbl[j]{2};
1294
                           if(sflag[h] == 1) outrns!(xx,yy,x,y);
1295
                           else outrns2(xx,yy,x,y);
1296
                           j=nbrplot;
1297
                        }
1298
                  }
1299
                  n++;
1300
              }
1301
               }
1302
1303
1304 return;
1305 }
1306
1307
1308 trnslink() {
          int flag, i, inp, j, k, l, m, n, on, outp, ptr, r, s, x, xx, v, yy;
1309
1310
1311
           ho2=file2:
           while(ptr=(hn2->trnotr)!=0) {
1312
          if(on=(ho2=>trnplot)==1) {
1313
1314
              x=(bp2=>xxcord);
1315
              y=(bo2->yvcord);
1316
              color(15);
              seamat(x,y=20,x,y+20);
1317
              1=(bp2->trnlen);
1318
1319
              k=(bp2->trnptr);
              for(i=0;i<1;i++) {
1320
```



```
Page 23 Fri Feb 8 05:01:00 1980
transgraph.c
                  charac((x-8+(8*i)),y+24,names[k+1]);
1321
1322
                  k++;
1323
              }
1324
1325
              color(14);
1326
              flag=0;
             - m=(bn2->intrn);
1327
1328
              r=iotbl(m);
1329
              for(i=0;i<r;i++) {
1330
                  inp=iotb1[m+1];
1331
                  for(j=0;j<nbrolot;j++) {</pre>
1332
                       if(inp==linktbl[j][0]) {
1333
                           xx=linktbl[j][1];
1334
                           yy=linktbl(j](2);
1335
                           if(flag==0) {
1336
                                    if(xx<=x) flag=1;
1337
                                    else flag=2;
1338
                           if(flag==1) intrns1(xx,yy,x,y);
1339
1340
                           else intrns2(xx,yy,x,y);
1341
                           j=nbrplot;
1342
                      }
1343
                  }
1344
                  m++;
           }
1345
1346
1347
              n=(bo2->outtrn);
1348
              s=iotbl[n];
1349
              for(i=0;i<s;i++) (
1350
                  outp=iotbl[n+1];
1351
                  for(j=0;j<nbrplot;j++) {</pre>
1352
                       if(outp==linktbl(j)(0)) {
1353
                           xx=linktbl[i][1];
1354
                           yy=linktb1[j][2];
1355
                           if(flag==1) outrnsl(xx,yy,x,y);
1356
                           else outrns2(xx,yy,x,y);
1357
                           j=nbrplot;
1358
                        }
1359
                  }
1360
                  n++;
1301
              }
1362
1303
          hp2++;
1364
1305 return;
1366 }
1367
1368
1369 intrnsl(xx,yy,x,y) {
1370
         int input;
1371
1372
          if(xx<(x-12)) (
1373
              if(yy < = y) {
                    if(yy==y) input=0;
1374
1375
                    else input=1;
1376
1377
              else inout=2;
1378
1379
              if(yy<=y) input=4;
1380
```



```
Page 24 Fri Feb 8 05:01:00 1980
transgraph.c
1381
             else inout=3;
1382
1383
1384
         switch (input) {
1385
             case 0:
1386
                  segmnt(xx+16,yy,x-1,y);
1387
                  rahtarrow(x-1,y);
1388
                  break;
1389
             case 1:
1390
                 segmnt(xx+12,yy+12,x-12,y-8);
1391
                  segmnt(x=12,y=8,x=1,y=8);
                  rahtarrow(x-1,y-8);
1392
1393
                  break;
1394
             case 2:
1395
                  segmnt(xx+12,yy-12,x-12,y+8);
1396
                  segmnt(x=12,y+8,x=1,y+8);
1397
                  rahtarrow(x=1,y+8);
1398
                  break;
1399
             case 3:
                  seamnt(xx-12,yy-12,x-12,y+36);
1400
1401
                  segmnt(x=12,y+16,x=1,y+16);
                  lftsemi(x-12,y+26);
1402
1403
                  rghtarrow(x-1,y+16);
1404
                  hreak;
             case 4:
1405
1406
                 segmnt(xx-12,yy+12,x-12,y-36);
                  seamnt(x-12,y-16,x-1,y-16);
1407
                  lftsemi(x+12,y-26);
1408
                  rghtarrow(x-1,y-16);
1409
1410
                  break;
1411
         - }
1412 return;
1413 }
1414
1415
1416 rghtarrow(x,v) {
1417
         segmnt(x=4,y=4,x,y);
1418
1419
         segmnt(x=4,y+4,x,y);
1420 return;
1421 }
1422
1423
1424 | ftsemi(x,v) {
1425
         int i,j,k;
1426
         double sart();
1427
         for(x=0;x<11;k++){
1428
             i = (v - (sart(100.-k*k)));
1429
              j=(v+(sart(100.-k*k)));
1430
             dot(x=(x/2.),511.-i);
1431
              not (x=(k/2.),511.-j);
1432
1433
              if(x>8) {
1434
                  dot(x=(k/2.),511.-i-2);
                  dot(x=(k/2.),511.-i-1);
1435
                  dot(x-(k/2.),511.-i+1);
1430
                  dot(x=(k/2.),511.-i+2);
1437
1438
                  dot(x=(k/2.),511.-j-2);
                  dot(x=(k/2.),511.-j=1);
1439
                  dot(x-(k/2.),511.-j+1);
1440
```



```
transgraph.c
                 Page 25
                            Fri Feb 8 05:01:00 1980
1441
                  dot(x-(k/2.),511.-j+2);
1442
              }
1443
         }
1444 return;
1445 }
1446
1447
1448 outrns1(xx,yy,x,y) {
1449
          int output;
1450
1451
         if(xx>(x+12)) {
1452
              if(yy \le y) {
1453
                  if(yy==y) output=0;
1454
                  else output=1;
1455
1456
              else output=2;
1457
1458
         else {
1459
              if(yy<=y) output=4;
1460
              else output=3;
1461
1462
1463
         switch (output) {
1464
              case 0:
1465
                  segmnt(x+1,y,xx-20,yy);
1466
                  rantarrow(xx-20,yy);
1467
                  break;
1468
              case 1:
1469
                  segmnt(x+12, y-8, xx-28, yy+8);
1470
                  seamnt(xx-28,yy+8,xx-20,yy+8);
                  rghtarrow(xx-20,yy+8);
1471
1472
                  segmnt(x+1,y-8,x+12,y-8);
1473
                  break;
1474
              case 2:
1475
                  segmnt(x+12, y+8, xx-28, yy-8);
                  segmnt(xx-28,yy-8,xx-20,yy-8);
1476
1477
                  rghtarrow(xx-20,yy-8);
                  seamnt(x+1,y+8,x+12,y+8);
1478
1479
                  break;
1480
              case 3:
1481
                  sermnt(x+1,y+16,x+12,y+16);
1482
                  seamnt (x+12,y+36,xx+28,yy=8);
                  segmnt(xx+28,yy-8,xx+20,yy-8);
1483
1484
                  lftarrow(xx+20,yy-8);
1485
                  rohtsemi(x+12,y+26);
1486
                  break;
1487
              case 4:
1488
                  segmnt (x+1,y-16,x+12,y-16);
1489
                  search(x+12,y-30,xx+23,yy+8);
1490
                  segmnt(xx+28,vy+8,xx+20,vy+8);
                  Iftarrow(xx+20,yv+8);
1491
                  rohtsemi(x+12,v-26);
1492
1493
                  break;
1494
1495 return;
1496 }
1497
1498
1499 rghtsemi(x,y) {
        int i,j,k;
1500
```



```
transgraph.c
                  Page 26
                            Fri Feb 8 05:01:00 1980
1501
          double sqrt();
1502
1503
          for (k=0;k<11;k++) {
              i = (y-(sart(100.-k+k)));
 1504
 1505
              j = (v+(sqrt(100.-k*k)));
1506
              dot(x+(k/2.),511.-i);
1507
              det(x+(k/2.),511.-j);
1508
              if(k>8) {
1509
                   dot(x+(k/2.),511.-i-2);
1510
                   dot(x+(k/2.),511.-i-1);
                   dot(x+(k/2.),511.-i+1);
1511
1512
                   dot(x+(k/2.),511.-i+2);
1513
                   dot(x+(k/2.),511.-j-2);
 1514
                   got(x+(k/2.),511.-j-1);
 1515
                   dot(x+(k/2.),511.-j+1);
                   dot(x+(k/2.),511.-j+2);
1516
1517
              }
 1518.
          }
 1519 return;
 1520 }
 1521
 1522
 1523 lftarrow(x,y) {
 1524
          segmnt(x+4,y=4,x,y);
1525
          segmnt(x+4,y+4,x,y);
1526
1527 return;
 1528 }
 1529
 1530
 1531 intrns2(xx, vy, x, v) {
 1532
          int input;
 1533
          if(xx>(x+12)) {
 1534
 1535
              if(yy \le y) {
1536
                   if(yy==y) input=0;
 1537
                   else inout=1;
 1538
 1539
              else innut=2;
1540
          }
 1541
          else {
              if(yy<=y) input=4;
 1542
              else input=3;
 1543
 1544
          }
 1545
 1546
          switch (input) {
              case 0:
 1547
                   seamnt(xx=16,yy,x+1,y);
 1548
                   lftarrow(x+1,y);
 1549
 1550
                   break;
 1551
              case 1:
                   seamnt(xx=12,yy+12,x+12,y=8);
 1552
                   seamnt(x+12,y=8,x+1,y=8);
 1553
                   lftarrow(x+1,y=8);
 1554
 1555
                   break;
 155c
               case 2:
                   searnt(xx-12,yy-12,x+12,y+8);
 1557
                   seamnt(x+12,y+8,x+1,y+8);
 1558
 1559
                   Iftarrow(x+1,y+8);
 1560
                   break;
```



```
transgraph.c
                 Page 27
                             Fri Feb 8 05:01:00 1980
1561
              case 3:
1502
                  seamnt(xx+12,yv-12,x+12,y+36);
1563
                  seamnt(x+12,y+16,x+1,y+16);
1564
                  rghtsemi(x+12,y+26);
1565
                  lftarrow(x+1,y+16);
1566
                  break;
1567
              case 4:
1568
                  segmnt(xx+12,yy+12,x+12,y-36);
1569
                  segmnt (x+12,v-16,x+1,y-16);
1570
                  rahtsemi(x+12,y-26);
1571
                  lftarrow(x+1,y=16);
1572
                  break;
         }
1573
1574 returni
1575 }
1576
1577
1578 outrns2(xx,yy,x,y) {
1579
         int output;
1580
1581
          if(xx<(x-12)) {
1582
              if(yy \le y) {
1583
                  if(yv==v) output=0;
1584
                  else outout=1;
1585
              }
1586
              else output=2;
1587
         }
1588
          else {
1589
             if(yy<=y) output=4;
1590
              else output=3;
1591
1592
          switch (output) {
1593
              case 0:
1594
1595
                  seamnt(x=1,y,xx+20,yy);
1596
                  lftarrow(xx+20,yy);
1597
                  break;
1598
              case 1:
                  segmnt(x=12,y=8,xx+28,yy+8);
1599
1600
                  segmnt(xx+28,yy+8,xx+20,yy+8);
1601
                  Iftarrow(xx+20,yy+8);
                  segmnt(x-1,y-8,x-12,y-3);
1602
1603
                  break;
1604
             case 2:
1605
                  seamnt(x-12,y+8,xx+28,yy-8);
                  segmnt(xx+28,yy-8,xx+20,yy-8);
1606
                  lfrarrow(xx+20,yy=8);
1607
1608
                  segmnt(x=1,y+8,x=12,y+8);
1609
                  break;
1610
              case 3:
                  seamnt(x=1,y+16,x=12,y+16);
1611
                  searnt(x=12,y+36,xx=28,yy=8);
1612
                  seamnt(xx-28,yy-8,xx-20,yy-8);
1613
1614
                  rghtarrow(xx-20,yy-8);
1615
                  lftsemi(x-12,y+26);
1616
                  break;
             case 4:
1617
                  seamnt(x=1,y=16,x=12,y=16);
1618
1619
                  segmnt (x-12,y-36,xx-28,yy+8);
                  segmnt(xx-28,yy+8,xx-20,yy+8);
1620
```



```
transgraph.c Page 28 Fri Feb 8 05:01:00 1980
1621
                  rghtarrow(xx-20,yy+8);
1622
                  lftsemi(x-12,y-26);
1623
                  break;
1624
1625 return;
1626 }
1627
1628
1629 coltab() {
1630 int i;
1631
1632
         i=11 *16;
1633
        lodcol(i++,15,15,15); /* color 0 */
        lodcol(i++,0,10,0); /* color 1 */
lodcol(i++,15,0,0); /* color 2 */
1634
                                   /* color 2 */
/* color 3 */
1635
1636
        logcol(i++,15,15,0);
                                   /* color 4 */
/* color 5 */
/* color 6 */
/* color 7 */
1637
         loacol(i++,12,0,12);
1638
         lodcol(i++,5,5,12);
1639
        lodcol(i++,6,6,5);
lodcol(i++,5,3,3);
1640
                                   /* color 8 */
1641
        lodcol(i++,10,0,10);
1642
        lodcol(i++,12,5,5);
                                   /* color 9 */
        lodcol(i++,5,5,3);
                                   /* color 10 */
1643
        lodcol(i++,12,12,0);
lodcol(i++,8,7,3);
1644
                                  /* color 11 */
                                   /* color 12 */
/* color 13 */
/* color 14 */
/* color 15 */
1645
         lodcol(i++,5,4,2);
1646
1646
1647
1648
         lodcol(i++,0,0,6);
         lodcol(i++,6,0,0);
1649 return;
1650 }
1651
1652
               /********************************
1653
               /******
1654
               /***** END OF PROGRAM TRANSGRAPH.C
1655
1656
1657
               /*****************************
1658
```



```
linkgraph.c Page 1 Fri Feb 8 07:09:45 1980
 2
 3
        5
                         PROGRAM LINKGRAPH.C
 7
        /******
 8
       /******
                    STEPHEN C. JENNINGS JC91 USMC
                      ROBERT J. HARTEL CS91 USA
 Q
       /******
10
       /******
1.1
                     WRITTEN FALL QUARTER 1979
12
       /******
                     NAVAL POSTGRADUATE SCHOOL
13
       /*****
                       MONTEREY, CALIFORNIA
       /******
1.4
15
       16
17
18
19 /*************
20 /*** EXTERNAL DECLARATIONS ****/
21 /*********************
22
23
24 /**** LITERALS ****/
25
26 #define header 4
27 #define std 10
28 #define fired 100
29 #define frames 150
                         /* contains control # as to bytes read in ... */
                         /* the standard input read buffer length .... */
                         /* the max # of transitions fired in 1 frame */
                         /* indicates the total # of network states .. */
30 #define bounds
                   360 · /* bounds on max # of nodes or transitions .. */
31 #define upper
                   2000
                        /* defines iotbl max length ...... */
                   3600
32 #define limit
                        33
34
35 /*** STRUCTURES ****/
36
37 struct (
                          /* data structure information on net nodes .. */
38
                    /* store control char not used in program ... */
/* index to names array ...... */
39
      int ctrll;
40
     int nameotr;
41
                     /* initial marker state of the network ..... */
     int marker;
42
                      /* x cordinate of place ..... */
     int xcord;
43
     int ycord;
                     /* y cordinate of place ..... */
44
      int olot;
                      /* whether or not place is to be plotted .... */
45
                      /* length of name associated with place .... */
     int length;
40
47 Ifilel [bounds], *hpl;
                         /* pointer into data structure ..... */
48
49
50 struct {
                         /* data structure information on transitions. */
51
52
     int ctrl2;
                     /* store control char not used in crogram ... */
     int trnotr;
53
                     /* index to names array ...... */
                      /* pointer to inputs for a transition ..... */
54
     int intrn;
55
                      /* pointer to outputs for a transition ..... */
     int outtrn;
56
                      /* x condinate of transition ..... */
     int xxcord;
57
     int vycord;
                     /* y cordinate of transition ..... */
58
     int trnplot:
                     /* whether or not transition is to be plotted */
59
     int trnlen;
                     /* length of name associated with transition */
60
```



```
Page 2 Fri Feb 8 07:09:45 1980
linkgraph.c
 61 )file2 (bounds), *DD2;
                            /* pointer into data structure ..... */
 62
 63
 64 /**** INTEGERS ****/
 ob int al,a2,a3,a4;
                             /* global storage for each data structure ... */
 67 int buffer(bounds);
                            /* buffer into which each frame is read .... */
 68 int cntrl[1];
                             /* variable containing # transitions fired .. */
                             /* keep track of nodes overflow status ..... */
 69 int ctroverflow;
 70 int dfltcolor;
                             /* a default color for indicating overflows . */
                             /* file descriptor for RUN--Y files ..... */
 71 int fdfbuf;
 72 int fdgbuf;
                             /* file descriptor for RUN--Z files ..... */
 73 int firing(bounds);
                             /* storage into which fired places are read . */
 74 int ictr;
                             /* counter passed to a function ..... */
 75 int ievents;
                             /* number of non- 3 displayable nodes ......
                                                                         * /
 76 int iflag:
                             /* counter for the interrupt mechanism ..... */
                             /* forms input-to-outout relationship ..... */
 77 int iotol(unperl;
 78 int kthframe;
                             /* counter for the iterations of the network. */
 79 int linktbl(bounds) (41;
                             /* version 1 & 2 screen nodes locations .... */
 80 int abrolot;
                             /* counter for number of displayed nodes .... */
 81
   int nbytes[2];
                             /* store count fields for data structures ... */
 82 int overflowtb1[fired1[2]; /* data structure to store overflow locations */
 83 int set;
                             /* user selected conrac graphics screen .....
                                                                         * /
 84 int tabcount;
                             /* counter for link revert ..... */
 95 int tblctr;
                             /* a counter for version 3..reset conditions. */
 86 int uniquetbl[fired][4];
                             /* reset table locations for version 3 ..... */
 87 int vers;
                             /* user selected option ......*/
                             /* coordinates retained for link revert .... */
 88 int xinstorelfired;
 39 int xoutstore[fired];
                             /* coordinates retained for link revert .... */
 90 int vinstore[fired];
                             /* coordinates retained for link revert .... */
                             /* coordinates retained for link revert .... */
 91 int youtstorelfired;
 95
 93
 94 /*** CHARACTERS ****/
 95
                             /* buffer store of file to be executed ..... */
 96 char couf[std];
97 char dbuf[std];
                            /* buffer store determining data scan ..... */
 98 char fbuf[std];
99 char qbuf[std];
                             /* buffer to store name of second file ..... */
                             /* buffer to store name of third file..... */
                             /* character array for node labels ..... */
100 char names[limit]:
101 char scrn;
                             /* option variable for display to the screen. */
                             /* buffer to store timing variable ..... */
102 char thuf[std];
103 cnar vbuf[std];
                            /* buffer to store version selected ..... */
104 char timing;
                            /* variable to set program execution timing . */
105
105
107 /******************
108 / * * * * FUNCTION MAIN * * * * * /
109 /******************
110
111
112 main() (
113
                         /* declare 'rubout' globally ..... */
114
       extern rubout();
115
       init();
                         /* read input file ...... */
                         /* verify if user wants to see data structure */
       determine();
116
      display();
                         /* display input to crt ..... */
117
                         /* select version of simulation & genisco set */
118
      select();
      prepare(2);
119
                         /* prepare genesco-conrac ...... */
                         /* draw network nodes on conrac ..... */
120
      drawnode();
```



```
Page 3
                       Fri Feb 8 07:09:45 1980
linkgraph.c
121
        places();
                           /* verify correct nodes drawn ..... */
122
       link();
                          /* function depicts network connectivity .... */
123
       imark();
                          /* starting status of network packets ..... */
       signal(2, rubout);
                          /* sets 'BRK' as interruot ..... */
124
125
                          /* successive iterations of network flow .... */
       marking();
126
       gnfini();
                          /* closing out graphics facilities ..... */
127
128 }
129
130
131 /*********************
132 /**** PROGRAM FUNCTIONS ****/
133 /***********************
134
135
136 pause(peroid) {
137
       /* function necessary as sleep() not compatible with signal() */
138
       int i,j,k;
139
       printf("***>>interrupt.....");
140
141
        for(i=0;i<oeroid;i++) {
           for(j=0;j<400;j++) {
142
143
                for(k=0;k<1000;k++) {
           1:
144
145
146
       }
147
       printf("***>>wait....");
148 return;
149 }
150
151
152 rubout() {
153
       /* function enables the 'brk' key as the interrupt signal */
154
       char halt;
155
156
       space(2);
       printf
157
158
        ("***>>> received signal...frame number %d...<ret> to continue \n",
       (iflag+2));
159
160
       printf
       ("***>>> for termination of program...type 'brk' from console \n");
161
       while ((halt=detchar())!='\n') {
162
163
             /* do-nothing loop */
164
       signal(2, rubout);
165
166 return;
167 }
168
169
170 space(returns) (
171
        int i;
172
         for (i=0; i<returns; i++) {</pre>
173
            printf("\n");
174
175
176 return;
177 }
178
179
180 init() {
```



```
linkgraph.c
                Page 4 Fri Feb 8 07:09:45 1980
181
        /* function opens unformatted file & initializes start condition */
182
        int a, bufctr, count, fd, i, j;
183
        char cif;
184
185
        space(2);
        printf("***->LINKGRAPH ILLUSTRATES NETWORK SIMULATION MODELS");
186
187
        space(2);
        printf("***->ENTER THE NAME OF THE FILE TO BE PROCESSED..BUT \n");
188
        printf("
                     MOTE THAT THIS FILE MUST BE AN UNFORMATTED FILE \n");
189
        printf("
190
                      PRODUCED AS A RESULT OF EXECUTING simulator.out \n");
191
        printf("
                       THE LAST LETTER OF WHICH MUST END IN LETTER 'X' \n");
192
      error:space(2);
193
        printf("***->");
194
195
         i=0;
196
         while((c=getchar()) != '\n') {
197
              cbuf[i]=c;
198
              1++;
199
         }
200
         cbuf[i]='\0';
201
202
         bufctr=i;
203
         for(j=0;j<bufctr;j++) {</pre>
             anuf[j]=fbuf[j]=cbuf[j];
204
             if(cbuf[j]=='X') (
205
206
                 fbuf[j] = 'Y';
207
                 abuf[j]='Z';
208
                 abuf [j+1] = touf [j+1] = '\0';
209
                 j=bufctr;
210
             }
211
212
213
        fd = open(cbuf,0);
214
        if (fd <= 0) {
215
             printf
             ("***->error occurred in opening file.....try again");
216
217
             space(3);
218
             goto error;
219
220
221
        if((count = read(fd,nbytes,header)) != header)
             printf("error occurred in nbytes read");
222
223
        ievents = (nbytes[1] - 1);
224
        al = nbytes[1];
225
        a = (nbytes[1]-1)+14;
226
        if((count = read(fd,file1,a)) != a)
             printf("error occured in file1 read");
227
228
229
        if((count=read(fd,obvtes,header))!=header)
230
            printf("error occurred in nbvtes read");
231
        a2=ntytes[1];
232
        a=(nbvtes(1)-1)*16;
233
        if((ccunt=read(fd,file2,a))!=a)
234
            printf("error occurred in file2 read");
235
236
        if((count=read(fd,nbytes,header))!=header)
237
            printf("error occurred in header read");
238
        a3=nbytes[1];
239
        a=(nbytes[1]+1)*2;
240
        if((count=read(fd,iotbl,a))!=a)
```



```
linkgraph.c
              Page 5 Fri Feb 8 07:09:45 1980
241
            printf("error occured in iotbl read");
242
243
        if((count=read(fd,nbytes,header))!=header)
            printf("error occurred in header read");
244
245
        a4=nbytes[1];
240
        a=nbytes[1]+1;
247
        if((count=read(fd,names,a))!=a)
248
            printf("error occurred in names read");
249
        close(fd);
250
251 return;
252 }
253
254
255 determine() {
256
        int i;
257
        char d;
258
259
        space(2);
        printf("***->FUNCTION 'DETERMINE' ALLOWS THE USER \n");
260
        printf(" TO EXAMINE ALL PRIMARY DATA STRUCTURES");
251
292
       over:space(2);
263
        printf
        ("***->IF THIS FEATURE IS DESIRED TYPE 1 IF NOT 0, THEN <RET>");
264
265
        space(2);
        printf("***->");
266
267
        i=0;
268
        while((d=metchar())!='\n') {
593
270
            dbuf[i]=d;
271
            1++;
272
        }
        dbuf[i]='\0';
273
274
275
        i = 0;
        while(dbuf[i]!='\0') {
276
277
            d=abuf[i];
278
            switch(d) {
                case'0':
279
                      scrn='0';
280
281
                      break;
282
                case'1':
                      scrn='1';
283
                      printf("***->USE CONTROL Q WHEN SCREEN FULL");
284
285
                      break;
286
                default:
                      orintf("***->either blank or invalid entry");
287
885
                      anto over;
289
                      break;
290
291
            i++;
505
        }
293 return;
294 }
295
296
297 display() {
298
       int i;
299
      if(scrn=='1') {
300
```



```
Fri Feb 8 07:09:45 1980
linkgraph.c
                Page 6
301
          scace(2);
302
          bol = file1;
303
          printf("***-> FILE1 DATA STRUCTURE");
304
          space(2);
305
          printf
          ("Infeed
306
                      #
                           marker
                                    . xcord
                                              ycord
                                                       plot
                                                             length \n");
307
          for (i=0;i<a1; i++){
308
               printf("%d \t %d \t %d \t %d \t %d \t %d \t %d \t \n",
309
                  bol->ctrl1,bol->nameptr, bol->marker,bpl->xcord,
310
                  bp1->ycord, po1->plot, pp1->length);
311
              bp1++;
312
          }
313
314
          space(2);
315
          bo2 = file2;
          printf("***-> FILE2 DATA STRUCTURE");
316
          space(2);
317
318
          printf
319
          ("Infeed
                                       outtrn xxcord yycord trnplot trnlen \n");
                      trnptr intrn
350
          for (i=0;1<a2; i++){
              orintf("%s \t %d \t \n",
321
                   bn2->ctrl2,bp2->trnptr,bp2->intrn,bp2->outtrn,
355
323
                   ho2->xxcord,bo2->yycord,bo2->trnplot,bp2->trnlen);
324
              502++;
325
          }
326
327
          space(2);
328
          printf("***-> IOTSL DATA ARRAY");
329
          space(2);
330
          for (i=0; i<a3; i++) {
              printf("%d ", iotbl[i]);
331
332
333
334
          space(2);
          orintf("***-> NAMES DATA ARPAY");
335
          space(2);
336
          for (i=0; i<a4+1; i++) {
337
             printf("%c", names[i]);
338
339
340
          space(2);
341
       }
342 return;
343 }
344
345
346 prepare(type) {
        /* function designates set, screen size and color table */
347
348
        int notovi
349
350
        y=0;
351
        aenisco (set);
352
        erase();
        screen(0.0,0.0,511.0,511.0);
353
354
        setmod(type);
355
        coltan();
356
        cclort(11);
357
        for(n=12;n<14;n++) {
358
            color(n);
             for(t=0;t<512;t++) {
359
                   segmnt(0, y+t,511, y+t);
360
```



```
linkgraph.c Page 7 Fri Feb 8 07:09:45 1980
361
362 }
363 return;
364 }
365
366
367 drawnode() {
368
        /* function displays type % location of network nodes */
369
        char c, *notr, hold;
370
        int a,b,clrlpl,count,d,entry,h,i,j,k,l,node,test,x,y,*z;
371
        float s;
372
        double sqrt();
373
374
        bol = filel;
        a=0;
375
        count = 1;
376
377
378
        while((test=(bp1->nameptr))!=0) {
379
            color(14);
380
            test++;
            z = &names[test];
381
382
            c = *z;
383
384
            if ((b=(bo1->plot))!= 0) {
385
                 switch (c){
                     case 'I':
386
387
                         node=1;
388
                         x = (hol -> xcord);
389
                         y = (bol -> ycord);
390
                         for(d=0;d<10;d++) {
391
                               segmnt(x=16,y+2+d,x,y+2+d);
392
                         if(vers==3) {
393
                              notr= &names(test+1);
394
395
                               hola= *nptr;
                               clrlbl=atoi(&hold);
396
                               label(x,y,test,clrlbl,node);
397
                               color(14);
398
399
                         }
400
                         else {
                               c1r1b1=14;
401
                               label(x,y,test,clrlbl,node);
402
403
                         linktbl(a)[3]=1;
404
405
                         break;
                     case '0':
400
                         node=2;
407
                         x = (ool \rightarrow xcord);
408
409
                         y = (hp1->ycord);
                         for(a=0;d<10;d++) {
410
                               segmnt(x=16,y=2=d,x,y=2=d);
411
412
                         if(vers==3) {
413
414
                              notr= %names[test+1];
                               hold= *notr;
415
                               clrlpl=atoi(&hold);
416
                               label(x,y,test,clrlbl,node);
417
418
                               color(14);
419
                         else {
420
```



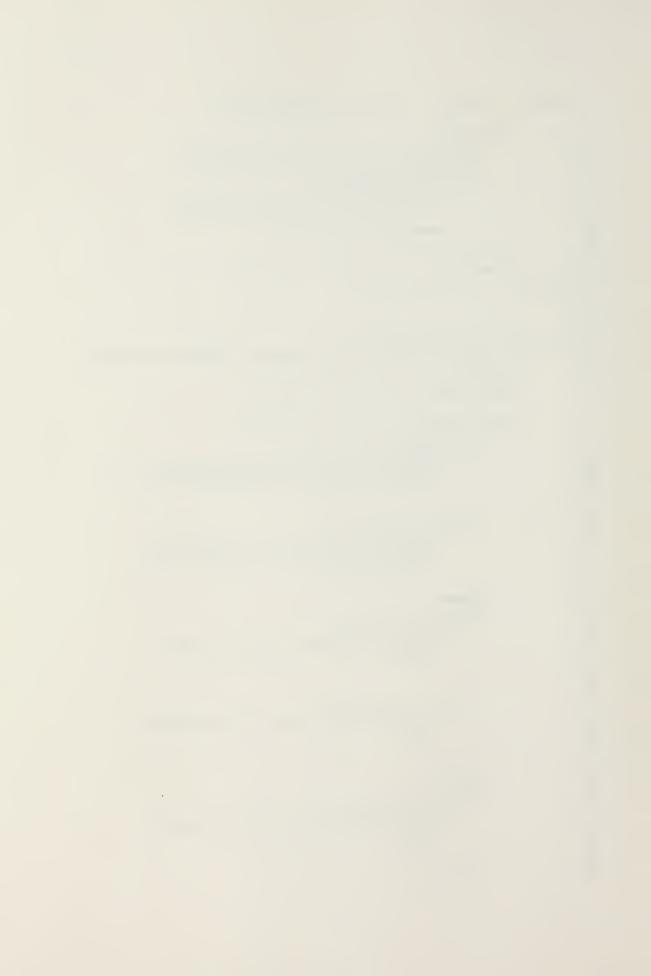
```
linkgraph.c Page 8 Fri Feb 8 07:09:45 1980
421
                               c1r1b1=14;
422
                               label(x,y,test,clrlbl,node);
423
424
                          linktbl[a1[3]=1;
425
                          break;
                      case 'R':
426
427
                          node=0;
428
                          x = (bol \rightarrow xcord);
429
                          y = (bc1->ycord);
430
                          for(d=0;a<31;d++) (
431
                          seamnt
432
                          (x-18+d/4,y+15-d,x+18-d/4,y+15-d);
433
434
                          clr1b1=14;
435
                          label(x,y,test,clrlbl,node);
436
                          break;
437
                     case 'S':
438
                          node=0;
439
                          x = (bp1->xcord);
                          v = '(hol->ycord);
440
441
                          for(d=0;d<31;d++) {
442
                            segmnt(x-18,y-15+d,x+18,y-15+d);
443
444
                          if(vers==3) (
                                notr= %names(test+1);
445
446
                                hold= *notr;
447
                                clrlbl=atoi(Shold);
448
                                lahel(x,y,test,clrlbl,node);
                                color(14);
449
450
451
                          else (
452
                                clr1b1=14;
453
                                label(x,v,test,clrlbl,node);
454
455
                          break;
                     case 'T':
456
                          node=0;
457
458
                          x = (bp1->xcord);
459
                          y = (pol->vcord);
                       · for (k=0;k<19;k++) {
460
461
                              s=k;
462
                              i = (x - (sart(324. - s*s)));
463
                              i=v+s;
464
                              1=(x+(surt(324.+s*s)));
465
                              seamnt(i,j,l,j);
466
467
                          for (k=0;k<19;k++) {
468
                              s=k;
469
                              i=(x-(sart(324.-s*s)));
470
                              j=v-s;
                              1=(x+(sart(324.-s*s)));
471
472
                              seamor(i,j,l,j);
473
                          if(vers==3) {
474
                               notr = %names[test+1];
475
                               hold = *nptr;
476
477
                               clrlbl=atoi(%hold);
                               label(x,v,test,clrlbl,nooe);
478
                               color(14);
479
480
```



```
linkgraph.c
                Page 9
                         Fri Feb 8 07:09:45 1980
481
                          else {
482
                               c1r1r1=14;
483
                               label(x,y,test,clrlbl,node);
484
485
                          break;
486
                     default:
487
                         printf
488
                          ("name not valid identifier");
489
                          space(2);
490
                          break;
491
                 -}
492
493
             linktbl(a)[0]=count;
494
             linkthl[a][1] = x;
495
             linktbl(a)(2) = y;
496
             a++;
497
498
        count++;
499
        bp1++;
500
501
         if(vers==3) (
502
             entry=0;
503
             uniquetol (entry) [0] = linktb1 [0] [0];
504
             uniquetbl [entry] [1] = linktbl [0] [1];
505
            uniquetol [entry] [2] = linktbl [0] [2];
506
            uniquetbl[entrv][3]=linktbl[0][3];
             rblctr=1;
507
508
             for(i=0;i<a;i++) {
509
                    510
                        uniquetb1 [entry] [2] == linktb1 [i+1] [2]) {
511
                        /* do nothing */
512
                    }
513
                    else {
514
                        entrv++;
                        uniquetbl(entrv)[0]=linktbl(i+1)[0];
515
516
                        uniquetbl [entry] [1] = linktbl [i+1] [1];
517
                        uniquetal(entry)(2)=linktbl(i+1)(2);
518
                        uniqueth! (entry! (3) = linktb! (i+1) (3);
519
                        tblctr++;
520
521
            }
522
523
        norclot=a;
524 return;
525 }
520
527
528 places() {
529
        int heigjeki
530
531
        if(scrn=='1') {
532
             space(2);
             orintf("***->DATA STRUCTURE LINKTBL ");
533
534
             space(2);
535
             for(i=0;i<nbrolot;i++) {</pre>
536
                 for(h=0;h<4;h++) {
                     printf("%a --",linktbl[i][h]);
537
538
539
                 space(1);
540
```



```
Page 10
                           Fri Feb 8 07:09:45 1980
linkgraph.c
541
             space(2);
542
             if(vers==3) {
543
                 printf("***->DATA STRUCTURE UNIQUETBL");
544
                 space(2);
545
                 for(j=0;j<tblctr;j++) {</pre>
546
                     for(k=0;k<4;k++) {
547
                          printf("%d --",uniquetbl(j)(k));
548
                     ŀ
549
                     space(1);
550
551
             }
552
             space(2);
553
        }
554 return;
555 }
556
557
558 label(xx,yv,zz,clol,olace) {
559
           /* determines node label placement in relation to node */
560
           int f,i;
561
562
           color(clbl);
563
564
            switch (place) {
565
                case 0:
566
                 if(xx>250) {
567
                    for(i=0;i<2;i++) {
568
                       cnarac((xx+(22+(8*i))),yy,names[zz]);
569
                        22++;
570
                    }
571
                 }
572
                 else {
573
                    for(i=0;i<2;i++) {
574
                       charac((xx-(34-(8*i))),yy,names[zz]);
575
                       ZZ++;
576
                    }
577
578
                 breaki
579
                case 1 :
                 if(xx>250) {
580
581
                    for(i=0;i<2;i++) {
582
                       charac((xx+4+(8*i)),yy+2,names(zz);
583
                       22++;
584
                    }
585
                 - }
586
                 else {
587
                    for(i=0;i<2;i++) {
588
                       cnarac((xx+(32+(3+i))),vy+2,names[zz]);
589
                       22++;
                    }
590
591
592
                 break;
593
                case 2 :
                 if(xx>250) {
594
                    for(i=0;i<2;i++) {
595
                       charac((xx+4+(8*i)),yv-10,names[zz]);
596
597
                       22++;
598
                 )
599
                 else (
600
```



```
linkgraph.c
                Page 11
                             Fri Feb 8 07:09:45 1980
                     for(i=0;i<2;i++) {
601
602
                        charac((xx-(32-(8*i))),yy-10,names[zz]);
603
                        22++;
604
605
606
                  break;
607
608 return;
609 }
610
611
612 link() {
        /* function links nodes by information stored in iotbl */
613
614
         int a.c.in[40],out[40],plotin[40],plotout[40];
615
         int content, ktr, i, k, m, maxinctr, maxoutctr, l, p, inx, iny, outx, outy;
616
        color(14);
617
618
        content=iotb1[1];
619
         ktr=1;
620
         while(content!=0) {'
621
             maxinctr=intol(ktr);
             for(i=0;i<content;i++) {</pre>
625
623
                 ktr++;
624
                  in[i]=iotbl [ktr];
625
             ktr++;
626
             content=iothl(ktr);
627
628
             maxoutctr=iotbl [ktrl;
629
             for(i=0;i<content;i++) {
630
                 ktr++;
631
                 out[i]=iotbl[ktr];
632
633
             for(n=0;p<maxinctr;p++) {</pre>
634
                 clotinic1=0;
635
                  for(!=";!<nornlot;!++) {</pre>
                      if(in[o] == linktb1[1][0]) {
636
637
                          nlotinlol=1;
638
                           1=nbrplot;
639
                 }
640
641
             }
642
643
             for(p=0;o<maxoutctr;p++) {
644
                 plotout(o)=0;
                  +or(1=0;1<nprp1et;1++) {
645
                      if(cut[o]==linktb][[][0]) {
646
                          clotout[o]=1;
647
648
                           innersiat;
649
                      ł
650
                 }
651
652
653
             for(k=0;k<maxinctr;k++) {
                  for(i=0;:<maxoutctr;i++) {
654
                      if(olotin[k] == 1 &% clotout[i] == 1) {
655
                          a=in[k];
656
                          b=out[il;
657
                           for(1=0;1<nhrplot;1++) {</pre>
658
                               if(linktb1[1][0]==a) {
659
                                   inx=linktbl[]][1];
660
```



```
linkgraph.c
                Page 12
                             Fri Feb 8 07:09:45 1980
661
                                    iny=linktb1[1][2];
662
                                    1=nhrolot;
663
664
665
                           for(m=0;m<nbrolot;m++) {</pre>
666
                               if(linktbl[m][0]==b) {
                                    outx=linkthl[m][1];
667
668
                                    outy=linktbl(m)(2);
669
                                    m=nbrplot;
670
                               }
671
672
                           lines(inx,iny,outx,outy);
673
674
                   }
675
             }
676
         ktr++;
677
        content=iotbl(ktr);
678
679 return;
680 }
681
682
683 lines(x1, v1, x2, y2) {
684
        int Inkcase, x, y;
685
686
         if(x1<=x2) {
687
             if(y1<=y2) {
                 if(x1==x2) lnkcase=3;
688
689
                  else (
690
                        if(y1==y2) lnkcase=0;
                        else Inkcase=4;
691
692
                 }
693
             }
694
             else {
                 if(x1==x2) lnkcase=1;
695
696
                 else inkcase=5;
697
             }
698
        }
699
         else {
700
             if(y1<=v2) {
701
                 if(v1==v2) lnkcase=2;
702
                 else Inkcase=7;
703
704
             else inkcase=b;
705
        switch(Inroase) {
700
707
             case 0:
                 Seamint(x1+20, v1, x2-20, y2);
708
709
                 x = x 5 - 50;
                 v=v2;
710
                 searnt(x-4, v-4, x, y);
711
712
                 secont (x=4,y+4,x,y);
713
                 oreak;
714
             case 1:
                 segrat(x1,y1=20,x2,y2+20);
715
716
                 x=x2;
                 v=v2+20;
717
                 searnt(x=4,v+4,x,y);
718
                 seament (x+4,y+4,x,y);
719
720
                 breaki
```



```
linkgraph.c
                Page 13
                           Fri Feb 8 07:09:45 1980
721
             case 2:
722
                  seamor(x1-20,y1,x2+20,y2);
723
                  x = x2 + 20;
724
                 y=y2;
725
                 seamnt(x+4,y-4,x,y);
726
                 seamnt(x+4,y+4,x,y);
727
                 breaki
728
             case 3:
729
                 segmnt(x1,y1+20,x2,v2=20);
730
                 x=x2;
731
                 y = y2 - 20;
732
                 seamnt(x-4,y-4,x,y);
733
                 seamnt(x+4,y-4,x,y);
734
                 break;
735
             case 4:
736
                 seamnt(x1+24,y1+24,x2-24,y2-24);
737
                 x=x2-10;
738
                 v=v2-16;
739
                 seamnt(x,y=6,x,y);
740
                 segmnt(x-6,y,x,y);
741
                 seamnt(x2-24,y2-24,x,y);
742
                 break;
743
             case 5:
744
                 segmnt(x1+24,y1-24,x2-24,y2+24);
745
                 x = x2 - 16;
746
                 v=v2+10;
747
                 segrat(x,y+6,x,y);
748
                 searnt(x=6,v,x,y);
749
                 segmnt(x2=24,y2+24,x,y);
750
                 break;
751
             case 6:
752
                 seamnt(x1-24,y1-24,x2+24,y2+24);
753
                 x=x2+16;
754
                 y = y2 + 16;
755
                 seamnt(x,y+6,x,y);
756
                 seamnt(x+b,y,x,y);
757
                 seamnt (x2+24, y2+24, x,y);
758
                 treak;
759
             case 7:
760
                 searnt(x1-24, v1+24, x2+24, y2-24);
761
                 x=x2+16;
762
                 v=y2-16;
703
                 seamnt(x,y=6,x,y);
764
                 seamnt(x+b,v,x,y);
765
                 searat(x2+24, v2-24, x, v);
760
                 breaki
757
        }
758 return;
769 1
770
771
772 select() {
773
        int in;
774
        char v;
775
776
        space(2);
        orintf("***->THERE ARE 3 VERSIONS TO THIS GRAPHICS PACKAGE \n");
777
        printf("
778
                   PLEASE SELECT ONE OF THE FOLLOWING VERSIONS: \n");
779
      again:space(2);
780
        printf("
                       VERSION 1 ... PETRI-NET PACKAGE ...... TYPE 1 \n");
```



```
linkgraph.c Page 14 Fri Feb & 07:09:45 1980
781
        printf("
                       VERSION 2 ... PACKET REPRESENTATION ... TYPE 2 \n");
        erintf("
                       VERSION 3 ... MULTIROUTING PACKAGE .... TYPE 3 \n");
782
783
        n = 0;
784
      twice:space(2);
785
        printf("***->");
786
        n++;
787
        i = 0;
788
        while ((v=getchar()) != '\n') (
789
            vbuf[i] = v;
790
            1++;
791
792
        vbuf[i] = '\0';
793
794
        i = 0;
795
        while(vbuf[i] != '\0') {
796
            v = vbuf[i];
797
            if(n==1) {
798
                   switch(v) {
                     case '1':
799
                      vers = 1;
800
801
                      break;
                     case '2':
802
803
                      vers = 2;
804
                      break;
                     case '3':
805
806
                      vers = 3;
807
                      break;
808
                     default:
809
                      printf
                      ("***->incorrect version try again!");
810
811
                      goto again;
812
                      break;
813
814
                   1++;
815
            }
816
            else {
817
                   switch(v) {
                     case '0':
818
819
                      set = 0;
820
                      break;
                     case '1':
821
                      set = 1;
822
823
                      break;
824
                     case '2':
                      set = 2;
825
826
                      nreak;
827
                     default:
828
                      printf
                      ("***">incorrect genisco set try again");
829
830
                      printf
                      ("
                             set selection should be 0,1,or2");
831
832
                      n = 1;
833
                      goto talce;
834
                      break:
835
                   }
                   1++;
836
837
            }
838
839
        if(n==1) (
840
             space(2);
```



```
linkgraph.c
                Page 15
                            Fri Feb 8 07:09:45 1980
841
              orintf("***->NOW SELECT THE GENISCO SET YOU WISH \n");
842
              printf("
                           THE PROGRAM TO BE DISPLAYED TO .... \n");
843
              orintf("
                            IN C3 LAB EITHER SETO, SET1 OR SET2 \n");
844
              goto twice;
845
846 return;
847 }
848
849
850 imark() {
851
        /\star marks initial state of system by calling appropriate function \star/
852
        int p,colour,e,g,x,y;
853
85.4
        bp1 = file1;
855
        afltcolor=3;
856
        color(afltcolor);
857
        colour=2;
858
        ctroverflow=0;
859
        while ((g=(bp1->nameptr)) !=0) {
860
            if((b=(bb1->plot)) !=0) {
861
                    switch(vers) {
862
                        case 1:
863
                            ivers1();
864
                           nreak;
865
                        case 2:
866
                            ivers2(colour);
867
                            break;
868
                        case 3:
869
                           ivers3(colour);
870
                           bréak;
871
                    }
872
873
            if(dfltcolor!=3) color(3);
874
            001++;
875
876
        color(14);
877
        printg(0,350.,480.,"TIME FRAME = 1");
878
        preread(1);
879
        hilite ();
880
        displa();
881
        hold();
882
        reset(1);
883
        color(13);
884
        ovrflow();
885
        print; (0,350.,480., "TIME FRAME = 1");
886 return;
887 }
888
889
890 hold() {
891
        int i;
892
893
       time:space(2);
        printf("***+>THIS IS THE INITIAL STATE OF THE NETWORK \n");
894
        printf("
895
                     ENTER THE TIMING MODE FOR EXECUTION .... \n");
        printf("
                      'O' FOR DU DELAYS ..'1' FOR FRAME PAUSES NO");
896
        printf("***->");
897
898
        i = 0;
899
        while((timing=getchar()) != '\n') {
900
            thuf[i] = timing;
```



```
linkgraph.c Page 16 Fri Feb 8 07:09:45 1980
901
            1++;
902
903
        tbuf(i) = '\0';
904
905
        i = 0;
906
        while(thuf(i) != '\0') {
907
            timino = thuf(i);
908
             i++;
909
910
        if(timing!='0' && timing!='1') {
911
            printf("***=>incorrect version try again!");
912
            goto time;
913
914 return;
915 }
916
917
918 ivers1() {
919
        int e,x,y,z;
920
        char check;
921
922
        e=(tp1+>marker);
923
        x = (hpl -> xcord);
924
        (trosy<-101);
925
        z=(rp1->namentr);
        if((check=names(z+1))!='I' && (check=names(z+1))!='0') {
926
927
             orinta(0,x-3.0,511.-(y-3),"%d",e);
928
929
        else {
930
             if(check=names(z+1)=='I') printg(0,x-14.0,511.+(v+2),"%d",e);
931
             else printq(0,x+14.0,511.+(y-9),"%d",e);
932
933 return;
934 }
935
936
937 ivers2(colour) (
938
        int e,x,y,z;
939
        char check;
940
941
        e=(bol->rarker);
942
        x=(bo1->xcord);
943
        v=(no1->ycord);
944
        z=(bo1->namestr);
945
        if((check=names(z+1))!='I' && (check=names(z+1))!='0') {
             ockt2(x,y,e,colour);
945
947
948
        else (
             if(cneck=namus(z+1)=='1') printg(0,x-14.0,511.-(y+2),"%d",e);
949
             else prints(0,x-14.0,5)1.-(y-9),"%p",e);
950
951
952 return;
953 }
954
955
956 ivers3(colour) {
957
        char keep, *kptr;
958
        int a,aa,b,po,c,cc,clr[fired],i,k,n,stack,total,x,y;
959
960
        total=0;
```



```
linkgraph.c
              Page 17
                          Fri Feb 3 07:09:45 1980
961
        0=1;
962
        x=(bp1->xcord);
963
        y=(bol->ycord);
964
        a=(bol->length);
965
        b=(bb1->nameptr);
966
        c=(bo1->marker);
967
        total=total+c;
968
969
        if(c>0) {
970
             for(i=0;i<c;i++) {
971
                 kptr = &names(b+a-2);
972
                 keep = *kotr;
973
                 clr[n] = atoi(&keep);
974
                 n++;
975
            }
976
        }
977
978
        kptr = %names(b+a);
        keep = *kotr;
979
980
        stack=atoi(8keep);
981
982
        if(names[b+(a-1)]!='0') {
983
            if(names(n+(a-1))=='2')
                                      stack=stack+20;
984
            else stack = stack + 10;
985
986
987
        for(i=0;i<stack-1;i++) {
988
            bo1++;
989
            aa=(bol->length);
990
            bo=(bol->nameotr);
991
            cc=(pol=>marker);
992
            total=total+cc;
993
             if(cc>0)
994
                 for(k=0;k<cc;k++)
                                    - {
995
                     kptr = &names(bb+aa-2);
996
                     keep = *kptr;
997
                     clr[n] =atoi(&keeo);
998
                     n++i
999
                }
1000
             }
1001
         if(names[o+1]!='I' && names[b+1]!='0') {
1002
1003
           pckt3
           (x,v,total,c1r[1],c1r[2],c1r[3],c1r[4],c1r[5],c1r[6],c1r[7],
1004
1005
            (fruotes
1000
1007
         else ( ·
             if(names(n+1)=='I') printo(0,x-14.0,511.-(y+2),"%d",total);
1008
1009
             else printa(0,x-14.0,511.-(y-9),"%d",total);
1010
1011 return;
1012 }
1013
1014
1015 preread(flan) {
          int bucket [2], count, fd, fa, i, nbrtrns;
1010
1017
          if(flac!=3) {
1018
         if(flag==1) {
1019
             fd = open (fbuf, 0);
1020
              if (fd<=0) {
```



```
Page 18
                          Fri Feb 8 07:09:45 1980
linkgraph.c
1021
                 printf("***->error occurred in opening fd file");
1022
             }
1023
              fdfbuf=fd;
1024
              fa = open (abuf, 0);
1025
              if (fg<=0) {
1026
                 printf("***->error occurred in opening fg file");
1027
             3
1028
             fdabuf=fa;
1029
1030
         if((count=read (fdfbuf, bucket, 2))!=2) {
             printf("***=>error occurred in fo bucket read");
1031
1032
1033
         if((count=read (fdfouf, buffer,(ievents*2)))!=(ievents*2)) {
1034
             printf("***=>error occurred in huffer read");
1035
1036
         if((count=read (fdapuf, bucket, 2))!=2) {
1037
             printf("***->error occurred in fq bucket read");
1038
1039
         if((count=read (fdopuf, cntrl,2))!=2) {
             printf("***->error occurred in cntrl read");
1040
1041
1042
         if(cntr1[0]==0) {
1043
             space(2);
1044
             printf("***->the last network state has been achieved");
1045
             kthframe=frames+1;
1046
             space(2);
1047
         }
1048
         else {
1049
             if((count=read (fdqbuf, bucket, 2))!=2) {
1050
                    printf("***->error occurred in bucket read");
1051
1052
             nbrtrns = cntrl(0) *2;
1053
             if((count=read (fddpuf, firing,nbrtrns)):= nbrtrns) {
                   printf("***=>error occurred in firing read");
1054
1055
1056
         }
1057
          }
1058
         else (
1059
         close(fdfbuf);
1060
         close(fdabuf);
1061
1062 return;
1063 }
1064
1065
1066 hilite() {
         int g,i,in[25],intpl,j,k,l,m,maxinctr,maxoutctr,n,out[25],
1067
1068
             outtol,p.plotin[25],plotout[25],inx,iny,qutx,outy;
1069
1070
         tabcount=0;
         tor(i=0;i<cntr1(01;i++) {
1071
1072
             bb2=file2;
             for(j=0;j<firin=[i]-1;j++) {
1073
                 502++;
1074
1075
             intbl=(nn2->intrn);
1075
1077
             outtpl=(bp2->outtrn);
             maxinctr=iothl[intbl];
1078
1079
             for(k=0;k<maxinctr;k++) {
1080
                 intbl++;
```

164



```
linkgraph.c
                Page 19
                             Fri Feb 8 07:09:45 1980
1081
                   in[k]=iotbl(intbl);
1082
              }
1083
              maxoutctr=iotbl [outtbl];
1084
              for(1=0;1<maxoutctr;1++) {</pre>
1085
                   cuttbl++;
1086
                   out[]]=iotbl[outtbl];
1087
1088
              for(p=0;p<maxinctr;p++) {
1089
                   plotin(p)=0;
1090
                   for(1=0;1<nbrolot;1++) {
1091
                       if(in[p] == linktbl[]][0]) {
1092
                            plotin[p]=1;
1093
                            1=nbrplot;
1094
                       }
1095
                   }
1096
1097
1098
              for(p=0;p<maxoutctr;p++) {</pre>
1099
                   olotout(p)=0;
1100
                   for(1=0;1<nbrelot;1++) {
1101
                       if(out(p) == linktbl(l](0)) {
1102
                            ; [= [a] tuotola
1103
                            l=mbrplot;
1104
1105
                   )
1106
1107
              for(k=0;k<maxinctr;k++) {
1108
                   for(g=0;g<maxoutctr;g++) {</pre>
                       if(plotin(k) == 1 && plotout(g) == 1) {
1109
1110
                            for(m=0;m<nbrolot;m++) {</pre>
1111
                                if(linktbl[m][0] == in[k]) {
1112
                                     inx=linktbl(m][1];
1113
                                     iny=linktbl[m][2];
1114
                                     m=nbrplot;
1115
                                }
1116
                            for(n=0;n<nbrolot;n++) {
1117
                                if(linktbl[n][0]==out[g]){
1118
                                     outx=linktbl(n)[1];
1119
1120
                                     outv=linktbl(n)(2);
1121
                                     n=nprolot;
                                }
1122
1123
1124
                           color(11);
1125
                            lines(inx,inv,outx,outy);
                            xinstpre[tapcount]=inx;
1125
                            viostore(tancount)=iny;
1127
                            xourstone (tabcount) = outx;
1126
1129
                            vourstbreltar countl=outv;
1130
                            tarcount++;
1131
                       }
1132
                    }
1133
               }
1134
         }
1135 return;
1136 }
1137
1138
1139 Inkryt() (
1140
         int i, inx, inv, outx, outy;
```



```
linkgraph.c
              Page 20
                          Fri Feb 8 07:09:45 1980
1141
1142
         color(14);
         for(i=v;i<tabcount;i++) {</pre>
1143
1144
              inx= xinstore[i];
1145
              iny= yinstore[i];
1146
              outx= xoutstore[i];
1147
              outy= youtstore[i];
1148
              lines(inx, inv, outx, outy);
1149
         }
1150 return;
1151 }
1152
1153
1154 stage() {
1155
1156
         if (kthframe!=0 && kthframe<frames+1) {
1157
             preread(2);
1158
              if(kthframe!=frames+1) {
1159
                  hilite();
1160
                  if(timing=='1') {
1161
                      disola();
1162
                      pause(1);
1163
1164
             }
1165
1166 return;
1167 }
1168
1169
1170 marking() {
1171
         /* function displays successive iterations of the network */
1172
         int colour, draw, i, mark, n, x, y;
1173
1174
         bol = filel;
         n = 2;
1175
1176
1177
         /* following loop processes levent # data segments each pass */
1178
         for(kthframe=0;kthframe<frames;kthframe++) {</pre>
1179
             stage();
1180
             if(kthframe>0) {
1181
                  reset();
1182
                  color(13);
1183
                  cvrflow();
1184
                  printa(0,350.,480.,"TIME FRAME = %d",n);
1185
                  n++;
1186
             }
1187
             iflan = +thirate;
             draw = (rel->plot);
1186
1189
             dfltcolor=3;
1190
             color (dilicolor);
1191
             colour=?;
1192
             ctroverflow=0;
1193
              for (i=0; i < ievents; i++) {
                  if (draw == 1) {
1194
1195
                      ictr = i;
1196
                      switch (vers) {
1197
                          case 1:
1198
                               vers1();
1199
                              preak;
                          case 2:
1200
```



```
linkgraph.c
               Page 21
                          Fri Feb 8 07:09:45 1980
1201
                               vers2(colour);
1202
                               break;
1203
                          case 3:
1204
                               vers3(colour);
1205
                               break;
1206
                      }
1207
                  }
1208
                  i = ictr++;
                  if(dfltcolor!=3) color(3);
1209
1210
1211
                  draw = (bp1 -> plot);
1212
              }
1213
              dfltcolor=14;
1214
             color(afltcolor);
1215
              colour=13;
1216
             printa(0,350.,480.,"TIME FRAME = %d",n);
1217
              if(timina=='1') oause(2);
1218
              Inkrvt();
1219
             bol = file1;
1220
         }
1221
         preread(3);
1222 return;
1223 }
1224
1225
1226 ovrflow() {
1227
1228
         int i,x,y;
1229
1230
         i=0;
1231
         while(overflowtb1(i1(01!=0) {
1232
             x=overflowrb1(i1(0);
1233
              y=overflowtb1[i][1];
1234
              if(x>250) {
1235
             plock((x+3A)*1.,511.-(y+10)*1.,(x+54)*1.,511.-y*1.);
1236
1237
             else (
1238
             block((x-50)*1.,511.-(y+10)*1.,(x-35)*1.,511.-y*1.);
1239
1240
             overflowtbl[i][0]=0;
             overflowth1(i)[11=0;
1241
1242
              1++;
1243
         }
1244 return;
1245 }
1246
1247
1248 vers1() {
1249
         int e,x,v,z;
1250
         char check;
1251
1252
         e=bufferlictrl;
1253
         :(racox<-fod)=x
         y=(bol=>vcord);
1254
1255
         z=(nol->nameptr);
1256
         if((check=names[z+11)!="1" && (check=names[z+11)!="0") {
1257
              printo(0,x-3.0,511.-(y-3),"%d",e);
1258
1259
         else {
1260
              if(check=names\{z+1\}=='I') printq(0,x-14.0,511.-(y+2),"%d",e);
```



```
linkgraph.c
              Page 22 Fri Feb 8 07:09:45 1980
1261
               else printa(0,x-14.0,511.-(y-9),"%d",e);
        }
1262
1263 return;
1264 }
1265
1266
1267 vers2(colour) {
1268
         int mark, x, y, z;
1269
         char check;
1270
1271
         x=(bo1->xcord);
1272
         y=(bp1->ycora);
1273
         z=(bp1->nameptr);
1274
         mark=buffer[ictr];
1275
         if((check=names{z+1})!='I' && (check=names{z+1})!='0') {
1276
          pckt2(x,y,mark,colour);
1277
1278
         else (
1279
          if((check=names{z+11)=='I') printg(0,x-14.0,511.-(y+2),"%d",mark);
1280
          else printq(0,x-14.0,511.-(y-9),"%d",mark);
1281
1282 return;
1283 }
1284
1285
1286 vers3(colour) {
         int a,aa,b,nb,mark,marks,clr[fired],j,k,n,stack,total,x,y;
1287
1288
         char keep, *kptr;
1289
1290
         total=0;
1291
         n=1;
1292
         x=(bp1->xcord);
1293
         y=(bo1->ycord);
1294
         a=(bp1->length);
1295
         b=(bp1->nameptr);
1296
         mark=buffer[ictr];
1297
         total=rotal+mark;
1298
         if(mark>0)
                     - {
1299
             for(j=0;j<mark;j++) {</pre>
1300
                  kotr = %names[b+a+2];
                  keeo = *kptr;
1301
1302
                 clr[n] = atoi(&keep);
1303
                 n++;
1304
             }
1305
         }
1306
1307
         kptr = %names[b+a];
1308
         keep = *kptr;
1309
         stack=atoi(&keep);
1310
1311
         if(names{b+(a-1)]!='0') {
             if(names(b+(a-1)) == '1') stack=stack+10;
1312
1313
             else (
                    if(names{b+(a-1)]=='2') stack=stack+20;
1314
                    else stack = stack + 30;
1315
1316
             - }
1317
1318
         for(j=0;j<stack-1;j++) {</pre>
1319
             bp1++;
1320
             ictr++;
```



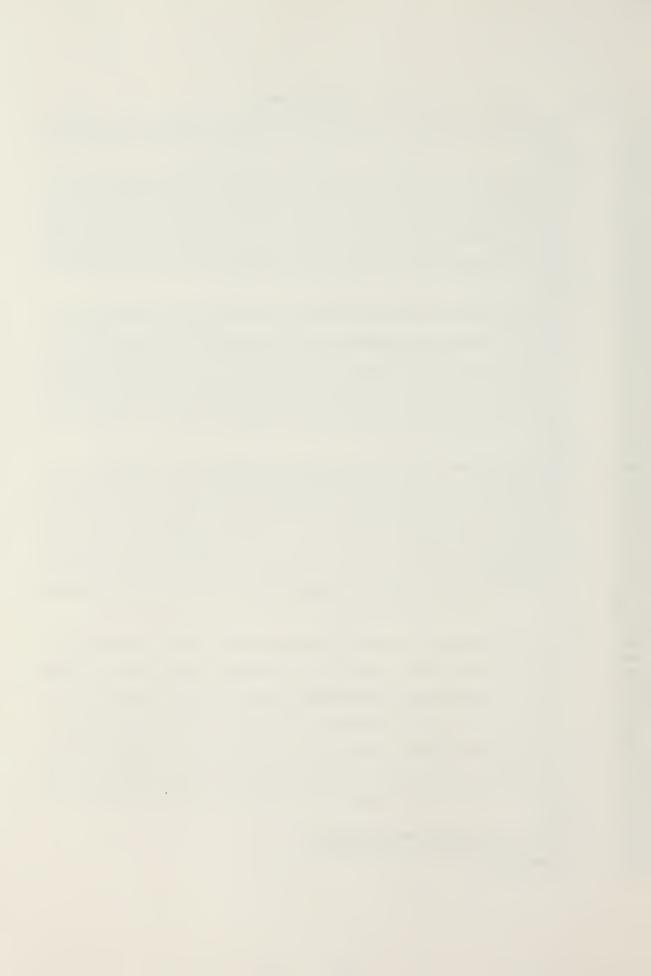
```
linkgraph.c
                Page 23.
                          Fri Feb 8 07:09:45 1980
1321
              aa=(bpl->length);
1322
              bb=(bbl->nameptr);
1323
              marks=puffer[ictr];
1324
              total=total+marks;
1325
              if(marks>0)
1326
                  for(k=0;k<marks;k++)
1327
                      kptr = &names[bb+aa-2];
1328
                      keep = *kptr;
1329
                      clr[n] =atoi(%keep);
1330
                      n++;
1331
                  }
1332
              1
1333
         }
1334
          if(names(b+1)!='I' && names(b+1)!='0') {
1335
1336
            (x,y,total,c|r[1],c|r[2],c|r[3],c|r[4],c|r[5],c|r[6],c|r[7],
1337
           colour);
1338
1339
         else {
1340
           color(3);
1341
           if(names(b+1) == 'I') printg(0,x-14.0,511.-(y+2), "%d", total);
1342
           else printa(0,x-14.0,511.-(y-9),"%d",total);
1343
1344
     return;
1345 }
1346
1347
1348 pckt2(xaxis,yaxis,point,class) {
1349
1350
      switch(point) {
1351
        case 0:
1352
1353
        case 1:
1354
         bloc<((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1355
         break:
1356
        case 2:
1357
         block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1358
         block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1359
         breaki
1360
        case 3:
1361
         plock((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1362
         block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
         block((xaxis-3)*1.,511.-(vaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1363
1364
         break;
1365
        case 4:
         nloc<((xaxis=3)*1.,511.=(yaxis+3)*1.,(xaxis+3)*1.,511.=(yaxis=3)*1.);
1366
1367
         block(fxaxis=3)+1.,511.=(vaxis=7)+1.,(xaxis+3)+1.,511.=(yaxis=13)+1.);
         plock((xaxis-3)*1.,511.-(vaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1368
1369
         olock((xaxis+7)*1.,511.-(vaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
1370
         break;
1371
        case 5:
1372
         block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
         plock((xaxis=3)+1.,511.=(vaxis=7)+1.,(xaxis+3)+1.,511.=(yaxis=13)+1.);
1373
         block((xaxis-3)*1.,511.-(vaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1374
1375
         nlock((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
1376
         plock((xaxis-13)*1..511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
1377
         break;
1378
        case n:
1379
         block((xaxis=3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis=3)*1.);
1380
         block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
```



```
Page 24
linkgraph.c
                           Fri Feb 8 07:09:45 1980
1381
         block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1382
         block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
1383
         block((xaxis=13)*1.,511.-(yaxis+3)*1.,(xaxis=7)*1.,511.-(yaxis=3)*1.);
         olock((xaxis-13)*1.,511.-(yaxis+13)*1.,(xaxis-7)*1.,511.-(yaxis+7)*1.);
1384
1385
         break;
1386
        case 7:
         plock((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1387
         plock((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1388
1389
         block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1390
         block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
1391
         block((xaxis-13)*1.,511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
1392
         block((xaxis-13)*1.,511.-(yaxis+13)*1.,(xaxis-7)*1.,511.-(yaxis+7)*1.);
1393
         block((xaxis+7)*1.,511.-(yaxis+13)*1.,(xaxis+13)*1.,511.-(yaxis+7)*1.);
1394
         break;
1395
        default:
1396
         overflowtrl(ctroverflow)[0]=xaxis;
1397
         overflowtb1 (ctroverflow) (1) =yaxis;
1398
         ctroverflow++;
1399
         block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1400
         block((xaxis=3)*1.,511.-(yaxis=7)*1.,(xaxis+3)*1.,511.-(yaxis=13)*1.);
1401
         block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1402
         block((xaxis+7)*1.,511.-(vaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.)%;
1403
         block((xaxis=13)*1.,511.-(yaxis+3)*1.,(xaxis=7)*1.,511.-(yaxis=3)*1.);
1404
         block((xaxis=13)*1.,511.=(yaxis+13)*1.,(xaxis=7)*1.,511.=(yaxis+7)*1.);
1405
         plock((xaxis+7)*1.,511.-(yaxis+13)*1.,(xaxis+13)*1.,511.-(yaxis+7)*1.);
1406
         color(class);
         dfltcolor=class;
1407
1408
         if(xaxis>250) {
1409
              printg(0, xaxis+38.0,511.-yaxis, "%d", point=7);
1410
         }
1411
         else {
1412
              printg(0, xaxis-50.0,511.-yaxis, "%d", point-7);
1413
         }
1414
         break;
1415
       }
1416 return;
1417 }
1418
1419
1420
1421 pckt3(xaxis,yaxis,total,c1,c2,c3,c4,c5,c6,c7,class)
1422
1423
      switch (total)
1424
       case 0:
1425
         break;
1426
       case 1:
1427
         color(c1);
         block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1428
1429
         break;
1430
       case 2:
1431
         color(c1);
         block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1432
1433
         color(c2);
         plock((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1434
1435
         break;
1430
       case 3:
         color(c1);
1437
         block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1438
1439
         color(c2);
         block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1440
```



```
Page 25
linkgraph.c
                           Fri Feb 8 07:09:45 1980
1 4 4 1
         color(c3);
1442
         block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1443
         break;
1444
       case 4:
1445
         color(cl);
1446
         block((xaxis=3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis=3)*1.);
1447
         color(c2);
         block((xaxis=3)*1.,511.=(yaxis=7)*1.,(xaxis+3)*1.,511.=(yaxis=13)*1.);
1448
1449
         color(c3);
1450
         block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1451
         color(c4);
1452
         block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
1453
1454
       case 5:
1455
         color(cl);
1456
         block((xaxis=3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis=3)*1.);
1457
         color(c2);
1458
         block((xaxis=3)*1.,511.-(yaxis=7)*1.,(xaxis+3)*1.,511.-(yaxis=13)*1.);
1459
         color(c3);
1460
         block((xaxis=3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1461
         color(c4);
1462
         plock((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.)%
1463
         color(c5);
1464
         block((xaxis-13)*1.,511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
1465
         break;
1466
       case 6:
         color(c1);
1467
1468
         block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1469
         color(c2);
1470
         block((xaxis=3)*1.,511.=(yaxis=7)*1.,(xaxis+3)*1.,511.=(yaxis=13)*1.);
1471
         color(c3);
         block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1472
         color(c4);
1473
1474
         block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
1475
         color(c5);
         płock((xaxis=13)*1.,511.-(yaxis+3)*1.,(xaxis=7)*1.,511.-(yaxis=3)*1.);
1476
1477
         color(c6);
         block((xaxis-13)*1.,511.-(yaxis+13)*1.,(xaxis-7)*1.,511.-(yaxis+7)*1.);
1478
1479
         break;
       case 7:
1480
1481
         color(c1);
         block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1482
1483
         color(c2);
1484
         block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1485
         color(c3);
         block((xaxis-3)*1.,511.-(vaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1486
1487
         color(c4);
         block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
1488
1489
         color(c5);
         block((xaxis-13)*1.,511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
1490
1491
         color(c6);
         block((xaxis-13)*1.,511.-(yaxis+13)*1.,(xaxis-7)*1.,511.-(yaxis+7)*1.);
1492
1493
         color(c7);
         block((xaxis+7)*1.,511.-(yaxis+13)*1.,(xaxis+13)*1.,511.-(yaxis+7)*1.);
1494
1495
         break;
1496
       default:
         overflowtb1[ctroverflow][0]=xaxis;
1497
1498
         overflowtb1[ctroverflow][1]=yaxis;
         ctroverflow++;
1499
1500
         color(cl);
```



```
Fri Feb 8 07:09:45 1980
               Page 26
linkgraph.c
1501
         block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1502
         color(c2);
1503
         block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1504
         color(c3);
1505
         block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1500
         color(c4);
         block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
1507
1508
         color(c5);
         clock((xaxis=13)*1.,511.=(yaxis+3)*1.,(xaxis=7)*1.,511.=(yaxis=3)*1.);
1509
1510
         color(cb);
1511
         block((xaxis=13)*1.,511.=(yaxis+13)*1.,(xaxis=7)*1.,511.=(yaxis+7)*1.);
1512
         color(c7);
         block((xaxis+7)*1.,511.-(yaxis+13)*1.,(xaxis+13)*1.,511.-(yaxis+7)*1.);
1513
1514
         color(class);
1515
         ofltcolor=class;
1516
         if(xexis>250) (
              printa(0, xaxis+38.0,511.-yaxis, "%d", total-7);
1517
1518
         }
1519
         else (
              printg(0, xaxis=50.0,511.-vaxis, "%d", total=7);
1520
         }
1521
1522
         breaki
1523
        }
1524 return;
1525 }
1526
1527
1528 reset() {
         /* reset function for successive network iterations */
1529
1530
         int i, mark, x, y, z;
1531
         if(yers==1 !! vers==2) {
1532
             for(i=0;i<nbrplot;i++)
1533
1534
             color(14);
1535
             x=linktbl(i)(1);
             y=linktbl[i][2];
1536
             z=1inktb1[i][3];
1537
1538
             if(z==0) {
                 block((x-3)*1.,511.-(y-7)*1.,(x+3)*1.,511.-(y-13)*1.);
1539
                 block((x-13)*1.,511.-(y+13)*1.,(x+13)*1.,511.-(y-3)*1.);
1540
1541
             }
              else {
1542
                 block((x-16)*1.,511.-(y-2)*1.,x*1.,511.-(y-10)*1.);
1543
                 block((x-16)*1.,511.-(y+10)*1.,x*1.,511.-(y+2)*1.);
1544
             }
1545
1546
             }
1547
         }
1548
         else (
             for(i=U;i<tblctr-1;i++) {
1549
             color(14);
1550
1551
             x=uniquetb1(i)(1);
             y=uniqueth1[i][2];
1552
1553
             z=uniquetb1[i][3];
1554
              if(z==0) {
                 block((x-3)*1.,511.-(y-7)*1.,(x+3)*1.,511.-(y-13)*1.);
1555
                 block((x-13)*1.,511.-(y+13)*1.,(x+13)*1.,511.-(y-3)*1.);
1556
1557
             }
1558
             else (
                 block((x-16)*1.,511.-(y-2)*1.,x*1.,511.-(y-10)*1.);
1559
                 block((x-16)*1.,511.-(y+10)*1.,x*1.,511.-(y+2)*1.);
1560
```



```
linkgraph.c Page 27 Fri Feb 8 07:09:45 1980
1561
            }
1562
            }
1563
        }
1564 return;
1565 }
1560
1567
1568 coltab() {
1569
       int i;
1570
1571
        i=11+16;
1572
        lodcol(i++,15,15,15);
                               /* color 0 */
        lodcol(i++,0,10,0);
                                           */
1573
                               /*
                                  color 1
1574
        lodcol(i++,15,0,0);
                                   color 2
                               / *
                                           */
1575
                               /*
        lodcol(i++,15,15,0);
                                   color 3
                                           */
1570
                                   color 4
                                           */
        (S1,0,51,++i) [comol
                               / *
1577
                               /* color 5 */
        lodcol(i++,5,5,12);
        loacol(i++,6,6,5);
1578
                               /* color 6
                                           */
1579
                               /* color 7
        lodcol(i++,5,3,3);
1580
                               /* color 8
        lodcol(i++,10,6,10);
                                           */
1581
        lodcol(i++,12,5,5);
                               /* color 9
                                           */
                               /* color 10 */
1582
        lodcol(i++,5,5,3);
1583
        lodcol(i++,12,12,0);
                               / ±
                                   color 11 */
1584
                               / *
                                   color 12 */
        103col(i++,8,7,3);
                               /* color 13 */
1585
        lodcol(i++,5,4,2);
1586
        lodcol(i++,0,0,6);
                               /* color 14 */
                               /* color 15 */
1587
        lodcol(i++,6,0,0);
1588 returni
1589 }
1590
1591
1592
             /****************************
1593
             /******
1594
                         END OF PROGRAM LINKGRAPH.C
             /******
1595
             /******
1590
             /******************************
1597
```



## LIST OF REFERENCES

- 1. Abramson, Norman and Franklin F. Kuo, Computer-Communication Networks, Prentice-Hall, 1973.
- 2. Anderson, George A, "Computer Interconnection Structures: Taxonomy, Characteristics, and Examples", Computing Surveys, Vol. 7, No. 4, Dec 1975.
- 3. Awad, Flias M., Systems Analysis and Design, Richard D. Irwin, Inc., 1979.
- 4. Browne, J. C., "Performance Analysis and Evaluation: The Connection to Reality", Research Directions in Software Technology, The MIT PRESS, 1979.
- 5. Cox, L. A. Jr., "Performance Prediction of Computer Architectures Operating on Linear Mathematical Models", Ph.D. Thesis, Computer Science Tept.. UC Davis Report UCRL-52582 (Sept 28,1978).
- 6. Cypser, R. J., Communications Architectures for Distributed Systems, Addison-Wesley Publishing Company, 1978.
- 7. Davies, D. W., and others, Computer Networks and Their Protocol, John Wiley and Sons, 1979.
- 8. Pixon, F., "The Impact of Technology on the Future Tactical Air Control Systems", Air Command and Staff College, March 1977.
- 9. Frslow, Philip H. Jr., "Multiprocessor Organization-A Survey", Computing Surveys, Vol. 9, No. 1, March, 1977.
- 10. Hamming, R. W., "Why Do You Believe the Simulation is Relevant?", Simulation, Naval Post Graduate School, 1979.
- 11. Heitreyer, Constance L., and others, "A Survey of Packet Switching Techniques for Broadcast Media", NRL Report 8035, Naval Research Laboratory Washington D.C., Oct 12, 1776.
- 12. Hopper, Grace, M., "David and Goliath", Selected Computer Articles 78, Dept. of Defense Computer Institute, 1978.

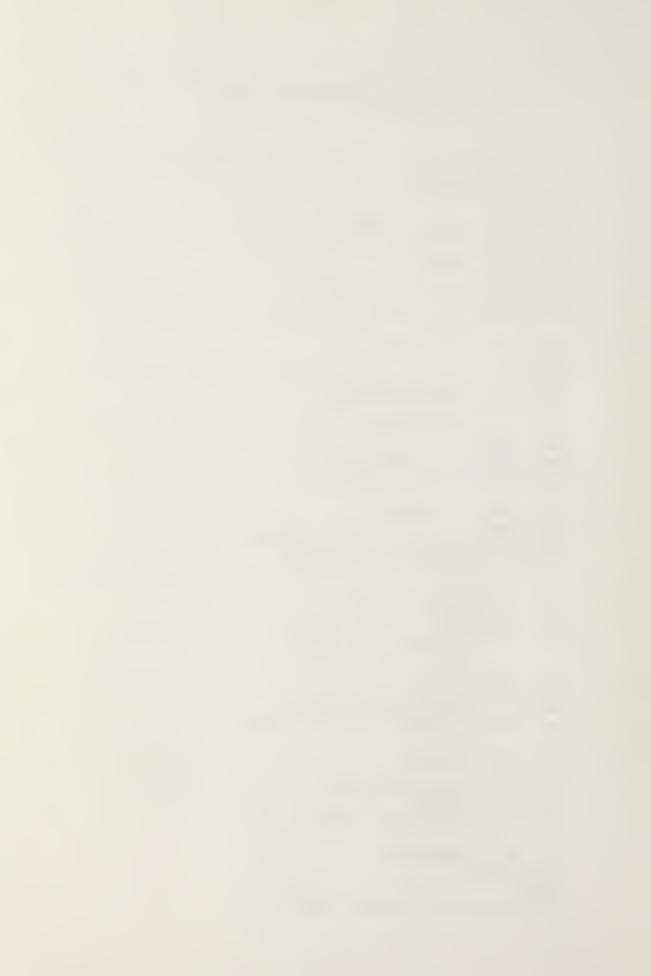


- 13. Kahn, Robert E., "The Organization of Computer Resources into a Packet Padio Network", IFEF Transactions on Communications, Vol. Com-25, No. 1, Jan 1977.
- 14. Kleinrock, Leonard, "Performance of Distributed Multi-Access Computer-Communication Systems", Proceedings of IFIP Congress 1977, Toronto, Canada, Aug 1977.
- 15. Kleinrock, Leonard, "Principles and Lessons in Packet Communications", Proceedings of the IFFE. Vol. 66, No. 11, Nov 1978.
- 16. Kunzelman, Ronald C., and others. "Packet Radio Experimental Network Research and Development, Final Technical Report", SPI International, Jan 1978.
- 17. Lawson, Billy R., "A Testbed for Evaluation of Packet Radio in an Army Tactical Corps", U.S.Army letter.
- 18. Moli, Gesualdo De, "On Networking", Computer Networks and Simulation, North-Holland, 1978.
- 19. Peterson, James L., "Petri-Nets", Computing Surveys, Vol. 9, No. 3, Sept 1977.
- 20. Price, W. L., Computer Networks and Simulation, North-Holland Publishing Company, 1978.
- 21. Stowers, Douglas M., "Computer Architecture Performance Prediction for Naval Fire Control Systems", Naval Post Graduate School Thesis, Dec 1979.
- 22. Terman, Lewis, M., "The Role of Microelectronics in Data Processing", Scientific American, Sept 1977.
- 23. Thurber, Kenneth J., and Leon D. Wald. "Associative and Parallel Processors", Computing Surveys. Vol. 7, No. 4, Dec 1975.
- 24. FM-24-1, Combat Communications, U.S. Army Field Manual.
- 25. Letter of Agreement (LOA) for PLRS/JTIDS Pybrid,
  U.S. Army Training and Toctrine Command. ACM 58401,
  Dept of the Army, 6 July 1979.

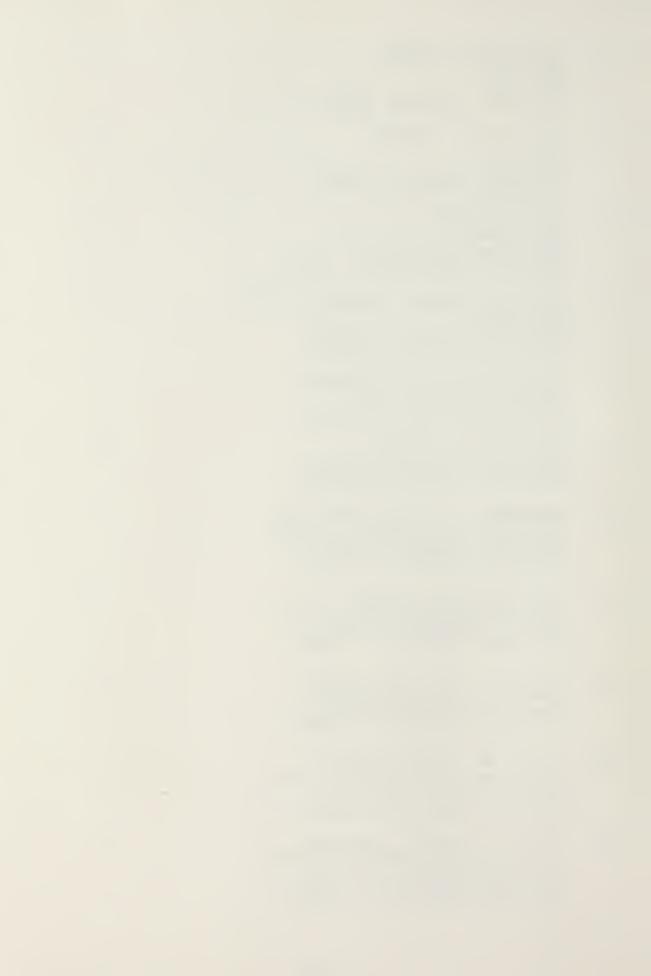


## INITIAL DISTRIBUTION LIST

		No.	Conies
1.	I. Larry Avrunin DTNSPDC Code 18 Bethesda, Maryland 20084		1
2.	Mr. P. P. Crabb Code 9134 Naval Ocean Systems Center San Diego, California 92152		1
3.	Mr. C. H. Gleissner DTNSRDC Code 18 Bethesda, Maryland 20084		1
4.	Ms. Kathryn Heninger Code 7503 Naval Research Laboratory Washington, D. C. 20375		1
5.	Mr. Roanld P. Kasik Code 4451 Naval Underwater Systems Center Newport, Rhode Island 02840		1
6.	Dr. J. McGraw U. C L.L.L. (L-794) P.O. Box 808 Livermore, California 94550		1
7.	Mr. Fenry G. Stuebing Code 503 Naval Air Development Center Warminster, Pennsylvannia 18974		1
8.	Mr. Joel Trimble Code 221 Office of Naval Research 800 North Guincy Arlington, Virginia 22217		1
9.	Mr. Mark Underwood Code P204 NPROC San Diego, California 92152		1



10.	Mr. Michael Wallace DTNSRIC Code 1828 Bethesda, Maryland 20084	1
11.	Mr. Walter P. Warner NSWC Code K70 Dahlgren, Virginia 22448	1
12.	Dr. John Zenor Code 31302 Naval Weapons Center China Lake, California 93555	1
13.	Office of Research Administration Code 012A Naval Postgraduate School Monterey, California 93940	1
14.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
15.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
16.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93940	2
17.	Prof. L. Cox, Code 52Cl Department of Computer Science Naval Postgraduate School Monterey, California 93940	5
18.	Prof. J. Wozencraft, Code 74 Chairman, C3 Academic Group Naval Postgraduate School Monterey, California 93942	2
19.	Prof. R. Schell, Code 52Sj Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
22.	Prof. O. Kodres, Code 52Kr Lepartment of Computer Science Naval Postgraduate School Monterey California 93942	1



21.	Cpt. Robert J. Hartel Code 110 Defense Communications Agency Washington D.C. 20005	5
22.	Capt. Stephen C. Jennings MCDEC Communication Officers School Quantico, Virginia 22134	2
23.	Mr. Pouglas M. Stowers Naval Sea Systems Command Code Sea 62Y21P Washington D.C. 20362	1







Thesis
J4726 Jennings
C.1 Petri-Net simulations
of communications networks.

8 NOV 83
104 25 85
12 SEP 86
12 MAY 88
32612

Thesis
J4726
Jennings
c.l Petri-Net simulations of communications net-

works.

thesJ4726
Petri-Net simulations of communications

3 2768 002 10750 0
DUDLEY KNOX LIBRARY